Graph Neural Networks (GNNs) – Mạng Lưới Neuron Đồ Thị

Nguyễn Quản Bá Hồng*

Ngày 12 tháng 9 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Graph Neural Networks (GNNs) – Mạng Lưới Neuron Đồ Thị.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/graph_neural_network/NQBH_graph_neural_network.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/graph_neural_network/NQBH_graph_neural_network.tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

1 Introduction to Graph Neural Networks

1.1 Keita Broadwater, Namid Stillmann. Graph Neural Networks in Action

- Fig: Mental model of GNN project. The steps involved for a GNN project are similar to many conventional ML pipelines, but we need to use graph-specific tools to create them. Start with raw data, which is then transformed into a graph data model & that can be stored in a graph database or used in a graph processing system. From the graph processing system (& some graph database), we can do exploratory data analysis & visualization. Finally, for graph ML, we preprocess data into a format that can be submitted for training & then train our graph ML model, in our examples, these will be GNNs. GNNs $\subset$ Graph ML models.

  – Hình: Mô hình tinh thần của dự án GNN. Các bước cần thực hiện cho 1 dự án GNN tương tự như nhiều quy trình ML thông thường, nhưng chúng ta cần sử dụng các công cụ dành riêng cho đồ thị để tạo ra chúng. Bắt đầu với dữ liệu thô, sau đó được chuyển đổi thành mô hình dữ liệu đồ thị & có thể được lưu trữ trong cơ sở dữ liệu đồ thị hoặc sử dụng trong hệ thống xử lý đồ thị. Từ hệ thống xử lý đồ thị (& 1 số cơ sở dữ liệu đồ thị), chúng ta có thể thực hiện phân tích dữ liệu thăm dò & trực quan hóa. Cuối cùng, đối với ML đồ thị, chúng ta xử lý trước dữ liệu thành 1 định dạng có thể được gửi để huấn luyện & sau đó huấn luyện mô hình ML đồ thị của chúng ta, trong các ví dụ của chúng ta, đây sẽ là các GNN. GNN $\subset$ Mô hình ML đồ thị.

- Foreword. Our world is highly rich in structure, comprising objects, their relations, & hierarchies. Sentences can be represented as sequences of words, maps can be broken down into streets & intersections, www connects websites via hyperlinks, & chemical compounds can be described by a set of atoms & their interactions. Despite prevalence of graph structures in our world, both traditional & even modern ML methods struggle to properly handle such rich structural information: ML conventionally expects fixed-sized vectors as inputs & is thus only applicable to simpler structures e.g. sequences or grids. Consequently, graph ML has long relied on labor-intensive & error-prone handcrafted feature engineering techniques. Graph neural networks (GNNs) finally revolutionize this paradigm by breaking up with regularity restriction of conventional DL techniques. They unlock ability to learn representations from raw graph data with exceptional performance & allow us to view DL as a much broader technique that can seamlessly generalize to complex & rich topological structures.

  – Thế giới của chúng ta vô cùng phong phú về cấu trúc, bao gồm các đối tượng, mối quan hệ của chúng, & hệ thống phân cấp. Câu có thể được biểu diễn dưới dạng chuỗi từ, bản đồ có thể được chia nhỏ thành các con phố & giao lộ, www kết nối các trang web thông qua siêu liên kết, & hợp chất hóa học có thể được mô tả bằng 1 tập hợp các nguyên tử & tương tác của chúng. Mặc dù cấu trúc đồ thị rất phổ biến trong thế giới của chúng ta, cả các phương pháp ML truyền thống & thậm chí hiện đại đều gặp khó khăn trong việc xử lý đúng cách thông tin cấu trúc phong phú như vậy: ML thường mong đợi các vectơ có kích thước cố định làm đầu vào & do đó chỉ áp dụng cho các cấu trúc đơn giản hơn, e.g. chuỗi hoặc lưới. Do đó, ML đồ thị từ lâu đã dựa vào các kỹ thuật thiết kế đặc trưng thủ công tốn nhiều công sức & dễ xảy ra lỗi. Mạng nơ-ron đồ thị (GNN) cuối cùng đã cách mạng hóa mô hình này bằng cách phá vỡ sự hạn chế về quy tắc của các kỹ thuật DL thông thường. Chúng mở khóa khả năng học các biểu diễn từ dữ liệu đồ thị thô với hiệu suất vượt trội & cho phép chúng ta xem DL như 1 kỹ thuật rộng hơn nhiều, có thể khái quát hóa liền mạch thành các cấu trúc tôpô phức tạp & phong phú.

  When Matthias Fey – creator of PyTorch Geometric & founding engineer Kumo.AI – begin to dive into field of graph ML, DL on graphs was still in its early stages. Over time, dozens to hundreds of different methods were developed, contributing incremental insights & refreshing ideas. Tools like our own PyTorch Geometric library have expanded significantly, offering cutting-edge graph-based building blocks, models, examples, & scalability solutions. Reflecting on this growth, it is clear how overwhelming it can be for newcomers to navigate essentials & best practices that have emerged over time, as valuable information is scattered across theoretical research papers or buried in implementations in GitHub repositories.

  – Khi Matthias Fey – người sáng lập PyTorch Geometric & kỹ sư sáng lập Kumo.AI – bắt đầu dấn thân vào lĩnh vực học máy đồ thị, học máy trên đồ thị vẫn còn ở giai đoạn sơ khai. Theo thời gian, hàng chục đến hàng trăm phương pháp khác

nhau đã được phát triển, đóng góp những hiểu biết sâu sắc & những ý tưởng mới mẻ. Các công cụ như thư viện PyTorch Geometric của chúng tôi đã mở rộng đáng kể, cung cấp các khối xây dựng, mô hình, e.g., & giải pháp khả năng mở rộng dựa trên đồ thị tiên tiến. Nhìn lại sự phát triển này, rõ ràng là những người mới bắt đầu có thể gặp khó khăn như thế nào khi tìm hiểu những điều cốt lõi & các phương pháp hay nhất đã xuất hiện theo thời gian, khi thông tin giá trị nằm rải rác trong các bài báo nghiên cứu lý thuyết hoặc bị chôn vùi trong các triển khai trên kho lưu trữ GitHub.

Now power of GNNs has been widely understood, this timely book provides a well-structured & easy-to-follow overview of field, providing answers to many pain points of graph ML practitioners. Hands-on approach, with practical code examples embedded directly within each chap, invaluably demystifies complexities, making concepts tangible & actionable. Despite success of GNNs across all kinds of domains in research, adoption in real-world applications remains limited to companies that have enough resources to acquire necessary knowledge for applying GNNs in practice. Confident: this book will serve as an invaluable resource to empower practitioners to over that gap & unlock full potentials of GNNs.

– Giờ đây, sức mạnh của GNN đã được hiểu rộng rãi, cuốn sách kịp thời này cung cấp 1 cái nhìn tổng quan được cấu trúc tốt & dễ hiểu về lĩnh vực này, giải đáp nhiều vấn đề khó khăn của các chuyên gia ML đồ thị. Phương pháp tiếp cận thực hành, với các ví dụ mã thực tế được nhúng trực tiếp trong mỗi chương, giúp làm sáng tỏ những điều phức tạp, biến các khái niệm thành hiện thực & khả thi. Mặc dù GNN đã thành công trong nhiều lĩnh vực nghiên cứu, việc áp dụng vào các ứng dụng thực tế vẫn chỉ giới hạn ở các công ty có đủ nguồn lực để có được kiến thức cần thiết cho việc áp dụng GNN vào thực tế. Tự tin: cuốn sách này sẽ là 1 nguồn tài nguyên vô giá giúp các chuyên gia vượt qua khoảng cách đó & khai phá toàn bộ tiềm năng của GNN.

- Preface. My journey into world of graphs began unexpectedly, during an interview at LinkedIn. As session wrapped up, shown a visualization of network – a mesmerizing structure that told stories without a single word. Organizations I had been part of appeared clustered, like constellations against a dark canvas. What surprised me most was that this structure was not built using metadata LinkedIn held about my connection; rather, it emerged organically from relationships between nodes & edges.

  – Hành trình khám phá thế giới đồ thị của tôi bắt đầu 1 cách bất ngờ, trong 1 buổi phỏng vấn tại LinkedIn. Khi buổi phỏng vấn kết thúc, 1 hình ảnh trực quan về mạng lưới được trình chiếu - 1 cấu trúc mê hoặc kể những câu chuyện mà không cần 1 lời nào. Các tổ chức mà tôi từng là thành viên hiện ra như những chòm sao trên nền vải tối. Điều khiến tôi ngạc nhiên nhất là cấu trúc này không được xây dựng bằng siêu dữ liệu mà LinkedIn nắm giữ về kết nối của tôi; thay vào đó, nó xuất hiện 1 cách tự nhiên từ các mối quan hệ giữa các nút & cạnh.

  Years later, driven by curiosity, I recreated that visualization. I marveled once again at how underlying connections along could map out an intricate picture of my professional life. This deepened my appreciation for power inherent in graphs – a fascination that only grew when I joined Cloudera & encountered graph neural networks (GNNs). Their potential for solving complex problems was captivating, but diving into them was like trying to navigate an uncharted forest without a map. There were no comprehensive resources tailored for nonacademics; progress was slow, often cobbled together from fragments & trial & error.

  – Nhiều năm sau, nhờ sự tò mò, tôi đã tái hiện lại hình ảnh đó. Tôi lại 1 lần nữa kinh ngạc trước cách các kết nối cơ bản có thể vẽ nên 1 bức tranh phức tạp về cuộc sống nghề nghiệp của mình. Điều này càng làm tôi trân trọng hơn sức mạnh tiềm ẩn của đồ thị – 1 niềm đam mê chỉ lớn dần khi tôi gia nhập Cloudera & gặp gỡ mạng nơ-ron đồ thị (GNN). Tiềm năng giải quyết các vấn đề phức tạp của chúng thật hấp dẫn, nhưng việc đào sâu vào chúng cũng giống như cố gắng khám phá 1 khu rừng chưa được khám phá mà không có bản đồ. Không có tài nguyên toàn diện nào được thiết kế riêng cho những người không chuyên; tiến độ rất chậm, thường được chắp vá từ những mảnh & thử & sai.

  This book is guide I wish I had during those early days. It aims to provide a clear & accessible path for practitioners, enthusiasts, & anyone looking to understand & apply GNNs without wading through endless academic papers or fragmented online searches. Hop: it serves as a 1-stop resource to learn fundamentals & paves way for deeper exploration. Whether you are here out of professional necessity, sheer curiosity, or same kind of amazement that 1st drew me in, invite to embark on this journey, bring potential of GNNs to life.

  – Cuốn sách này chính là cẩm nang mà tôi ước mình đã có trong những ngày đầu ấy. Nó hướng đến việc cung cấp 1 lộ trình rõ ràng & dễ tiếp cận cho các chuyên gia, người đam mê, & bất kỳ ai muốn hiểu & áp dụng GNN mà không cần phải lội qua vô số bài báo học thuật hay tìm kiếm trực tuyến rời rạc. Hop: nó là 1 nguồn tài nguyên tổng hợp để học các kiến thức cơ bản & mở đường cho những khám phá sâu hơn. Cho dù bạn đến đây vì nhu cầu công việc, sự tò mò đơn thuần, hay cùng 1 sự ngạc nhiên như đã thu hút tôi lần đầu, tham gia vào hành trình này, khai phá tiềm năng của GNN.

- About this book. GNNs in Action is a book designed for people to jump quickly into this new field & start building applications. At same time, try to strike a balance by including just enough critical theory to make this book as standalone as possible. Also fill in implementation details that may not be obvious or are left unexplained in currently available online tutorials & documents. In particular, information about new & emerging topics is very likely to be fragmented. This fragmentation adds friction when implementing & testing new technologies.

  – GNNs in Action là 1 cuốn sách được thiết kế để mọi người có thể nhanh chóng bước vào lĩnh vực mới này & bắt đầu xây dựng ứng dụng. Đồng thời, cố gắng cân bằng bằng cách đưa vào vừa đủ lý thuyết quan trọng để cuốn sách này trở nên độc lập nhất có thể. Đồng thời, bổ sung những chi tiết triển khai có thể chưa rõ ràng hoặc chưa được giải thích trong các hướng dẫn trực tuyến hiện có & tài liệu. Đặc biệt, thông tin về các chủ đề mới & đang nổi lên rất có thể sẽ bị phân mảnh. Sự phân mảnh này gây khó khăn khi triển khai & thử nghiệm các công nghệ mới.

With GNNs in Action, offer a book that can reduce that friction by filling in gaps & answering key questions whose answers are likely scattered over internet or not covered at all. Done so in a way that emphasizes approachability rather than high rigor.

– Với GNNs in Action, cung cấp 1 cuốn sách có thể giảm thiểu sự khó khăn đó bằng cách lấp đầy những khoảng trống & trả lời những câu hỏi quan trọng mà câu trả lời có thể nằm rải rác trên internet hoặc chưa được đề cập đến. Hãy làm điều này theo cách nhấn mạnh tính dễ tiếp cận hơn là tính nghiêm ngặt cao.

- Who should read this book. This book is designed for ML engineers & data scientists familiar with neural networks but new to graph learning. If have experience in OOP, find concepts particularly accessible & applicable.

  – Cuốn sách này được thiết kế dành cho các kỹ sư ML & nhà khoa học dữ liệu đã quen thuộc với mạng nơ-ron nhưng chưa quen với học đồ thị. Nếu có kinh nghiệm về OOP, tìm các khái niệm đặc biệt dễ hiểu & áp dụng.

- How this book is organized: A road map. In Part 1 of this book, provide a motivation for exploring GNNs, as well as cover fundamental concepts of graphs & graph-based ML. In Chap. 1, introduce concepts of graphs & graph ML, providing guidelines for their use & applications. Chap. 2 covers graph representations up to & including node embeddings. This will be 1st programmic exposure to GNNs, which are used to create such embeddings.

  – Trong Phần 1 của cuốn sách này, chúng tôi sẽ cung cấp động lực để khám phá GNN, cũng như đề cập đến các khái niệm cơ bản về đồ thị & Học máy dựa trên đồ thị. Trong Chương 1, chúng tôi sẽ giới thiệu các khái niệm về đồ thị & Học máy dựa trên đồ thị, đồng thời cung cấp hướng dẫn sử dụng & ứng dụng của chúng. Chương 2 sẽ đề cập đến các biểu diễn đồ thị, bao gồm cả nhúng nút. Đây sẽ là lần đầu tiên chúng tôi tiếp xúc với GNN, được sử dụng để tạo ra các nhúng như vậy.

  In part 2, core of book, introduce major types of GNNs, including graph convolutional networks (GCNs) & GraphSAGE in Chap. 3, graph attention networks (GATs) in Chap. 4, & graph autoencoders (GAEs) in Chap. 5. These methods are bread & butter for most GNN applications & also cover a range of other DL concepts e.g. convolution, attention, & autoencoders.

  – Trong phần 2, cốt lõi của cuốn sách, giới thiệu các loại GNN chính, bao gồm mạng tích chập đồ thị (GCN) & GraphSAGE trong Chương 3, mạng chú ý đồ thị (GAT) trong Chương 4, & bộ mã hóa tự động đồ thị (GAE) trong Chương 5. Các phương pháp này là nền tảng cho hầu hết các ứng dụng GNN & cũng bao gồm 1 loạt các khái niệm DL khác, e.g., tích chập, chú ý, & bộ mã hóa tự động.

  In part 3, look at more advanced topics. Describe GNNs for dynamic graphs (spatio-temporal GNNs) in Chap. 6 & give methods to train GNNs at scale in Chap. 7. Finally, end with some consideration for project & system planning for graph learning projects in Chap. 8.

  – Trong phần 3, xem xét các chủ đề nâng cao hơn. Mô tả GNN cho đồ thị động (GNN không gian-thời gian) trong Chương 6 & đưa ra các phương pháp huấn luyện GNN ở quy mô lớn trong Chương 7. Cuối cùng, kết thúc bằng 1 số cân nhắc về dự án & lập kế hoạch hệ thống cho các dự án học đồ thị trong Chương 8.

- About code. Python is coding language of choice throughout this book. There are now several GNN libraries in Python ecosystem, including PyTorch Geometric (PyG), Deep Graph Library (DGL), GraphScope, & Jraph. Focus on PyG, which is 1 of most popular & easy-to-use frameworks, written on top of PyTorch. Want this book to be approachable by an audience with a wide set of hardware constraints, so with exception of some individual sects & Chap. 7 on scalability, distributed systems & GPU systems aren't required, although they can be used for some of coded examples.

  – Python là ngôn ngữ lập trình được lựa chọn trong suốt cuốn sách này. Hiện nay, hệ sinh thái Python đã có 1 số thư viện GNN, bao gồm PyTorch Geometric (PyG), Thư viện Đồ thị Sâu (DGL), GraphScope, Jraph. Tập trung vào PyG, 1 trong những framework phổ biến nhất, dễ sử dụng, được viết trên nền tảng PyTorch. Tôi muốn cuốn sách này dễ tiếp cận với những độc giả có nhiều hạn chế về phần cứng, vì vậy, ngoại trừ 1 số điểm riêng biệt trong Chương 7 về khả năng mở rộng, các hệ thống phân tán & GPU không bắt buộc, mặc dù chúng có thể được sử dụng cho 1 số ví dụ được mã hóa.

  Book provides a survey of most relevant implementations of GNNs, including graph convolutional networks (GCNs), graph autoencoders (GAEs), graph attention networks (GATs), & graph long short-term memory (LSTM). Aim: cover GNN tasks mentioned earlier. In addition, touch on different types of graphs, including knowledge graphs.

  – Sách cung cấp 1 bản tổng quan về các triển khai GNN phổ biến nhất, bao gồm mạng tích chập đồ thị (GCN), bộ mã hóa tự động đồ thị (GAE), mạng chú ý đồ thị (GAT), bộ nhớ dài hạn đồ thị (LSTM). Mục tiêu: bao quát các nhiệm vụ GNN đã đề cập trước đó. Ngoài ra, đề cập đến các loại đồ thị khác nhau, bao gồm đồ thị tri thức.

  This book contains many examples of source code both in numbered listings & in line with normal text. In both case, source code is formatted in a `fixed-width font like this` to separate it from ordinary text. Sometimes code is also **in bold** to highlight code

PART 1: 1ST STEPS. Graphs are 1 of most versatile & powerful ways to represent complex, interconnected data. This 1st part introduces fundamental concepts of graph theory, explaining what graphs are, why they matter as a data type, & how their structure captures relationships that traditional data formats miss. Explore building blocks of graphs & different graph types.

– Đồ thị là 1 trong những phương pháp linh hoạt & mạnh mẽ nhất để biểu diễn dữ liệu phức tạp, có liên kết với nhau. Phần 1 này giới thiệu các khái niệm cơ bản của lý thuyết đồ thị, giải thích đồ thị là gì, tại sao chúng quan trọng như 1 kiểu dữ liệu, & cách cấu trúc của chúng nắm bắt các mối quan hệ mà các định dạng dữ liệu truyền thống bỏ sót. Khám phá các khối xây dựng của đồ thị & các kiểu đồ thị khác nhau.

Explore fundamental concepts about GNNs, beginning with what they are & how they differ from traditional neural networks. With this foundation, study graph embeddings, uncovering how to represent graphs in a way that makes them useful for ML.

These concepts set stage for mastering GNNs & their transformative capabilities in later chaps. By end of this book, have a solid understanding of basics, preparing you to dive deeper into mechanics of GNNs.

– Khám phá các khái niệm cơ bản về GNN, bắt đầu với bản chất của chúng & sự khác biệt so với mạng nơ-ron truyền thống. Với nền tảng này, nghiên cứu nhúng đồ thị, khám phá cách biểu diễn đồ thị sao cho hữu ích cho ML. Những khái niệm này đặt nền tảng cho việc nắm vững GNN & khả năng biến đổi của chúng trong các chương sau. Khi đọc xong cuốn sách này, bạn sẽ có được kiến thức cơ bản vững chắc, sẵn sàng cho việc tìm hiểu sâu hơn về cơ chế hoạt động của GNN.

- 1. Discovering graph neural networks. Covers:
  - Defining graphs & GNNs
  - Understanding why people are excited about GNNs
  - Recognizing when to use GNNs
  - Taking a big picture look at solving a problem with a GNN

For data practitioners, fields of ML & DS initially excite us because of potential to draw nonintuitive & useful insights from data. In particular, insights from ML & DL promise to enhance our understanding of world. For working engineer, these tools promise to deliver business value in unprecedented ways.

– Đối với các chuyên gia dữ liệu, lĩnh vực Học máy & Phân tích dữ liệu (ML & DS) ban đầu khiến chúng ta hào hứng vì tiềm năng rút ra những hiểu biết phi trực quan & hữu ích từ dữ liệu. Đặc biệt, những hiểu biết từ Học máy & Phân tích dữ liệu (ML & DL) hứa hẹn sẽ nâng cao hiểu biết của chúng ta về thế giới. Đối với các kỹ sư đang làm việc, những công cụ này hứa hẹn mang lại giá trị kinh doanh theo những cách chưa từng có.

Experience deviates from this ideal. Real-world data is usually messy, dirty, & biased. Furthermore, statistical methods & learning systems come with their own set of limitations. An essential role of practitoners: comprehend these limitations & bridge gap between real data & a feasible solution. E.g., may want to predict fraudulent activity in a bank, but 1st need to make sure that our training data has been correctly labeled. Even more importantly, need to check that our models won't incorrectly assign fraudulent activity to normal behaviors, possibly due to some hidden confounders in data.

– Kinh nghiệm thực tế thường khác xa lý tưởng này. Dữ liệu thực tế thường lộn xộn, bẩn thỉu, & thiên vị. Hơn nữa, các phương pháp thống kê & hệ thống học tập cũng có những hạn chế riêng. 1 vai trò thiết yếu của người thực hành: hiểu rõ những hạn chế này & thu hẹp khoảng cách giữa dữ liệu thực tế & 1 giải pháp khả thi. Ví dụ: có thể muốn dự đoán hoạt động gian lận trong 1 ngân hàng, nhưng trước tiên cần đảm bảo rằng dữ liệu đào tạo của chúng ta đã được dán nhãn chính xác. Quan trọng hơn nữa, cần kiểm tra xem các mô hình của chúng ta có gán sai hoạt động gian lận cho các hành vi bình thường hay không, có thể do 1 số yếu tố gây nhiễu tiềm ẩn trong dữ liệu.

For graph data, until recently, bridging this gap has been particularly challenging. Graphs are a data structure that is rich with information & especially adept at capturing intricacies of data where relationships play a crucial role. Graphs are omnipresent, with relationship data appearing in different forms e.g. atoms in molecules (nature), social networks (society), & even models connection of web pages on internet (technology). Important to note: term *relational* here does not refer to *relational databases*, but rather to data where relationships are of significance.

– Đối với dữ liệu đồ thị, cho đến gần đây, việc thu hẹp khoảng cách này đặc biệt khó khăn. Đồ thị là 1 cấu trúc dữ liệu giàu thông tin & đặc biệt khéo léo trong việc nắm bắt những dữ liệu phức tạp, nơi các mối quan hệ đóng vai trò then chốt. Đồ thị hiện diện ở khắp mọi nơi, với dữ liệu mối quan hệ xuất hiện dưới nhiều dạng khác nhau, e.g.: nguyên tử trong phân tử (tự nhiên), mạng xã hội (xã hội), & thậm chí cả mô hình kết nối các trang web trên internet (công nghệ). Điều quan trọng cần lưu ý: thuật ngữ *relational* ở đây không đề cập đến *relational databases*, mà là dữ liệu trong đó các mối quan hệ có ý nghĩa quan trọng.

Previously, if you wanted to incorporate relational features from a graph into a DL model, it had to be done in an indirect way, with different models used to process, analyze, & then use graph data. These separate models often couldn't be easily scaled & had trouble taking into account all node & edge properties of graph data. To make best use of this rich & ubiquitous data type for ML, needed a specialized ML technique specifically designed for distinct qualities of graphs & relational data. This is gap that GNNs fill.

– Trước đây, nếu muốn tích hợp các đặc điểm quan hệ từ đồ thị vào mô hình DL, việc này phải được thực hiện gián tiếp, sử dụng các mô hình khác nhau để xử lý, phân tích, & sau đó sử dụng dữ liệu đồ thị. Các mô hình riêng biệt này thường không dễ dàng mở rộng & gặp khó khăn trong việc tính đến tất cả các thuộc tính nút & cạnh của dữ liệu đồ thị. Để tận dụng tối đa kiểu dữ liệu phong phú & phổ biến này cho ML, cần có 1 kỹ thuật ML chuyên biệt được thiết kế riêng cho các đặc tính riêng biệt của đồ thị & dữ liệu quan hệ. Đây chính là khoảng trống mà GNN lấp đầy.

DP field often contains a lot of hype around new technologies & methods. However, GNNs are widely recognized as a genuine leap forward for graph-based learning [2]. This does not mean: GNNs are a silver bullet. Careful comparisons should be done between predictive results derived from GNNs & other ML & DL methods.

– Lĩnh vực DP thường chứa đựng nhiều thông tin cường điệu về các công nghệ & phương pháp mới. Tuy nhiên, GNN được công nhận rộng rãi là 1 bước tiến thực sự cho học tập dựa trên đồ thị [2]. Điều này không có nghĩa là: GNN là giải pháp hoàn hảo. Cần so sánh cẩn thận giữa các kết quả dự đoán thu được từ GNN & các phương pháp ML & DL khác.

Key thing to remember: if your DS problem involves data that can be structured as a graph – i.e., data is connected or relational – then GNNs could offer a valuable approach, even if you weren't aware that sth was missing in your approach.

GNNs can be designed to handle very large data, to scale, to adapt to graphs of different sizes & shapes. This can make working with relationship-centric data easier & more efficient, as well as yield richer results.

– Điều quan trọng cần nhớ: nếu bài toán DS của bạn liên quan đến dữ liệu có thể được cấu trúc dưới dạng đồ thị - i.e., dữ liệu được kết nối hoặc quan hệ - thì GNN có thể cung cấp 1 phương pháp tiếp cận hữu ích, ngay cả khi bạn không nhận ra rằng phương pháp của mình còn thiếu điều gì đó. GNN có thể được thiết kế để xử lý dữ liệu rất lớn, có khả năng mở rộng, thích ứng với các đồ thị có kích thước & hình dạng khác nhau. Điều này có thể giúp việc xử lý dữ liệu tập trung vào mối quan hệ dễ dàng & hiệu quả hơn, cũng như mang lại kết quả phong phú hơn.

Standout advantages of GNNs are why data scientists & engineers are increasingly recognizing importance of mastering them. GNNs have the ability to unveil unique insights from relational dat – from identifying new drug candidates to optimizing ETA prediction accuracy in your Google Maps app – acting as a catalyst for discovery & innovation, & empowering professionals to push boundaries of conventional data analysis. Their diverse applicability spans various fields, offering professionals a versatile tool that is as relevant in e-commerce (e.g., recommendation engines) as it is in bioinformatics (e.g., drug toxicity prediction). Proficiency in GNNs equips data professionals with a multifaceted tool for enhanced, accurate, & innovative data analysis of graphs.

– Những lợi thế nổi bật của GNN là lý do tại sao các nhà khoa học dữ liệu & kỹ sư ngày càng nhận thức được tầm quan trọng của việc thành thạo chúng. GNN có khả năng khám phá những hiểu biết độc đáo từ dữ liệu quan hệ – từ việc xác định các ứng cử viên thuốc mới đến tối ưu hóa độ chính xác dự đoán ETA trong ứng dụng Google Maps – đóng vai trò là chất xúc tác cho khám phá & đổi mới, & trao quyền cho các chuyên gia vượt qua các giới hạn của phân tích dữ liệu thông thường. Khả năng ứng dụng đa dạng của chúng trải rộng trên nhiều lĩnh vực, mang đến cho các chuyên gia 1 công cụ đa năng, vừa phù hợp trong thương mại điện tử (e.g.: công cụ đề xuất) vừa phù hợp trong tin sinh học (e.g.: dự đoán độc tính của thuốc). Thành thạo GNN trang bị cho các chuyên gia dữ liệu 1 công cụ đa năng để phân tích dữ liệu đồ thị chính xác, & sáng tạo.

For all these reasons, GNNs are now popular choice for recommender engines, analyzing social networks, detecting fraud, understanding how biomolecules behave, & many other practical examples.

– Vì tất cả những lý do này, GNN hiện là lựa chọn phổ biến cho các công cụ đề xuất, phân tích mạng xã hội, phát hiện gian lận, hiểu cách các phân tử sinh học hoạt động, & nhiều ví dụ thực tế khác.

- 1.1. Goals of this book. GNNs in Action is aimed at practitioners who want to begin to deploy GNNs to solve real problems. This could be a ML engineer not familiar with graph data structures, a data scientist who hasn't yet tried GNNs, or even a software engineer who may be unfamiliar with either. Throughout this book, cover topics from basics of graphs all way to more complex GNN models. Build up architecture of a GNN, step-by-step. This includes overall architecture of a GNN & critical aspect of message passing. Then go on to add different features & extensions to these basic aspects, e.g. introducing convolution & sampling, attention mechanisms, a generative model, & operating on dynamic graphs. When building our GNNs, work with Python & use some standard libraries. GNNs libraries are either standalone or use TensorFlow or PyTorch as a backend. In this text, focus will be on PyTorch Geometric (PyG). Other popular libraries include Deep Graph Library (DGL, a standalone library) & SPektral (which uses Keras & TensorFlow as a backend). There is also Jraph for JAX users.

  – GNNs in Action hướng đến những người thực hành muốn bắt đầu triển khai GNN để giải quyết các vấn đề thực tế. Họ có thể là 1 kỹ sư ML chưa quen thuộc với cấu trúc dữ liệu đồ thị, 1 nhà khoa học dữ liệu chưa từng thử GNN, hoặc thậm chí là 1 kỹ sư phần mềm chưa quen thuộc với cả hai. Xuyên suốt cuốn sách này, chúng tôi sẽ đề cập đến các chủ đề từ kiến thức cơ bản về đồ thị cho đến các mô hình GNN phức tạp hơn. Xây dựng kiến trúc của GNN, từng bước một. Điều này bao gồm kiến trúc tổng thể của GNN & khía cạnh quan trọng của việc truyền thông điệp. Sau đó, tiếp tục thêm các tính năng khác nhau & phần mở rộng cho các khía cạnh cơ bản này, e.g.: giới thiệu tích chập & lấy mẫu, cơ chế chú ý, mô hình sinh, & hoạt động trên đồ thị động. Khi xây dựng GNN, làm việc với Python & sử dụng 1 số thư viện chuẩn. Thư viện GNN có thể độc lập hoặc sử dụng TensorFlow hoặc PyTorch làm nền tảng. Trong văn bản này, trọng tâm sẽ là PyTorch Geometric (PyG). Các thư viện phổ biến khác bao gồm Thư viện Deep Graph (DGL, 1 thư viện độc lập) & SPektral (sử dụng Keras & TensorFlow làm nền tảng). Ngoài ra còn có Jraph dành cho người dùng JAX.

  Our aim throughout this book is to enable you to:

  1. access suitability of a GNN solution for your problem.
  2. understand when traditional neural networks won't perform as well as a GNN for graph structured data & when GNNs may not be the best tool for tabular data.
  3. design & implement a GNN architecture to solve problems specific to you.
  4. make clear limitations of GNNs.

  This book is weighted toward implementation using programming. Also devote some time on essential theory & concepts, so that techniques covered can be sufficiently understood. These are covered in an "Under Hood" sect at end of most chaps to separate technical reasons from actual implementation. There are many different models & packages that build on key concepts introduced in this book. So, this book should not be seen as a comprehensive review of all GNns methods & models, which could run to several thousands of pages, but rather starting point for curious & eager-to-learn practitioner.

  – Mục tiêu của chúng tôi trong suốt cuốn sách này là giúp bạn:

  1. đánh giá tính phù hợp của giải pháp GNN cho vấn đề của bạn.
  2. hiểu khi nào mạng nơ-ron truyền thống không hoạt động tốt bằng GNN đối với dữ liệu có cấu trúc đồ thị & khi nào GNN có thể không phải là công cụ tốt nhất cho dữ liệu dạng bảng.

3. thiết kế & triển khai kiến trúc GNN để giải quyết các vấn đề cụ thể của bạn.
4. làm rõ những hạn chế của GNN.

Cuốn sách này thiên về việc triển khai bằng lập trình. Đồng thời dành thời gian cho các lý thuyết & khái niệm thiết yếu, để các kỹ thuật được đề cập có thể được hiểu đầy đủ. Những điều này được trình bày trong phần "Under Hood" ở cuối hầu hết các chương để phân biệt lý do kỹ thuật với việc triển khai thực tế. Có rất nhiều mô hình & gói khác nhau được xây dựng dựa trên các khái niệm chính được giới thiệu trong cuốn sách này. Vì vậy, cuốn sách này không nên được coi là 1 bài tổng quan toàn diện về tất cả các phương pháp & mô hình GNN, có thể dài tới hàng nghìn trang, mà nên là điểm khởi đầu cho những người thực hành & ham học hỏi.

Book is divided into 3 parts. Part 1 covers basics of GNNs, especially ways in which they differ from other neural networks, e.g. *message passing & embeddings*, which have specific meaning for GNNs. Part 2, heart of book, goes over models themselves, where we cover a handful of key model types. Then, in part 3, go into more detail with some of harder models & concepts, including how to scale graphs & deal with temporal data.

– Sách được chia thành 3 phần. Phần 1 trình bày những kiến thức cơ bản về GNN, đặc biệt là những điểm khác biệt giữa chúng với các mạng nơ-ron khác, e.g. truyền thông điệp & nhúng, vốn có ý nghĩa riêng đối với GNN. Phần 2, trọng tâm của sách, sẽ đề cập đến bản thân các mô hình, trong đó chúng ta sẽ tìm hiểu 1 số loại mô hình chính. Sau đó, trong phần 3, chúng ta sẽ đi sâu hơn vào 1 số mô hình & khái niệm khó hơn, bao gồm cách chia tỷ lệ đồ thị & xử lý dữ liệu thời gian.

GNNs in Action is designed for people to jump quickly into this new field & start building applications. Aim for this book: reduce friction of implementing new technologies by filling in gaps & answering key development questions whose answers may not be easy to find or may not be covered elsewhere at all. Each method is introduced through an example application so you can understand how GNNs are applied in practice.

– GNNs in Action được thiết kế để mọi người nhanh chóng tiếp cận lĩnh vực mới này & bắt đầu xây dựng ứng dụng. Mục tiêu của cuốn sách này: giảm thiểu sự cản trở khi triển khai các công nghệ mới bằng cách lấp đầy những khoảng trống & trả lời những câu hỏi phát triển quan trọng mà câu trả lời có thể không dễ tìm hoặc chưa được đề cập ở bất kỳ nơi nào khác. Mỗi phương pháp được giới thiệu thông qua 1 ứng dụng ví dụ để bạn có thể hiểu cách GNN được áp dụng trong thực tế.

* 1.1.1. Catching up on graph fundamentals. Do need to understand basics of graphs before you can understand GNNs. Goal for this book is to teach GNNs to DL practitioners & builders for traditional neural networks who may not know much about graphs. At same time, also recognize: readers of this book may vary enormously in their knowledge of graphs. How to address these differences & make sure everyone has what they need to make the most of this book? In this chap, provide an introduction to fundamental graph concepts that are most essential to understanding GNNs.
  – Bạn cần nắm vững những kiến thức cơ bản về đồ thị trước khi có thể hiểu về GNN. Mục tiêu của cuốn sách này là hướng dẫn GNN cho những người thực hành DL & những người xây dựng mạng nơ-ron truyền thống, những người có thể chưa biết nhiều về đồ thị. Đồng thời, cũng cần lưu ý: kiến thức về đồ thị của độc giả có thể rất khác nhau. Làm thế nào để giải quyết những khác biệt này & đảm bảo mọi người đều có những kiến thức cần thiết để tận dụng tối đa cuốn sách này? Trong chương này, chúng tôi sẽ giới thiệu các khái niệm cơ bản về đồ thị, những khái niệm thiết yếu nhất để hiểu về GNN.
  After refresher on key concepts in graphs & graph learning, look into some case studies in several fields where GNNs are being successfully applied. Then, break down those specific cases to see what makes a good case for using a GNN, as well as how to know if you have a GNN problem on your hands. At end of chap, introduce mechanics of GNNs, barebone skeleton that the rest of book will add to.
  – Sau khi ôn lại các khái niệm chính về đồ thị & học đồ thị, xem xét 1 số nghiên cứu điển hình trong 1 số lĩnh vực mà GNN đang được ứng dụng thành công. Sau đó, phân tích các trường hợp cụ thể đó để xem đâu là lý do tốt để sử dụng GNN, cũng như cách nhận biết liệu bạn có đang gặp vấn đề về GNN hay không. Cuối chương, giới thiệu cơ chế hoạt động của GNN, bộ khung xương cốt mà phần còn lại của cuốn sách sẽ bổ sung.

◦ 1.2. Graph-based learning. This section defines graphs, graph-based learning, & some fundamentals of GNNs, including basic structure of a graph & a taxonomy of different types of graphs. Then, review graph-based learning, putting GNNs in context with other learning methods. Finally, explain value of graphs, ending with an example of data derived from Titanic dataset.

– Học tập dựa trên đồ thị. Phần này định nghĩa đồ thị, học tập dựa trên đồ thị, & 1 số kiến thức cơ bản về mạng nơ-ron nhân tạo (GNN), bao gồm cấu trúc cơ bản của đồ thị & phân loại các loại đồ thị khác nhau. Sau đó, xem xét lại học tập dựa trên đồ thị, đặt GNN vào bối cảnh của các phương pháp học tập khác. Cuối cùng, giải thích giá trị của đồ thị, kết thúc bằng 1 ví dụ về dữ liệu được lấy từ tập dữ liệu Titanic.

* 1.2.1. What are graphs? Graphs are data structures with elements, expressed as *nodes or vertices*, & relationships between elements, expressed as *edges or links*. All nodes in graph will have additional *feature data*. This is node-specific data, relating to things e.g. names or ages of individuals in a social network. Links are key to power of relational data, as they allow us to learn more about system, give new tools for analyzing data, & predict new properties from it. This is in contrast to tabular data e.g. a database table, dataframe, or spreadsheet, where data is fixed in rows & columns.
  – Đồ thị là cấu trúc dữ liệu với các phần tử, được biểu diễn dưới dạng *nút hoặc đỉnh*, & mối quan hệ giữa các phần tử, được biểu diễn dưới dạng *cạnh hoặc liên kết*. Tất cả các nút trong đồ thị sẽ có thêm *dữ liệu đặc trưng*. Đây là dữ liệu cụ thể của từng nút, liên quan đến các thông tin như tên hoặc tuổi của các cá nhân trong mạng xã hội. Liên kết là chìa khóa cho sức mạnh của dữ liệu quan hệ, vì chúng cho phép chúng ta tìm hiểu thêm về hệ thống, cung cấp các công cụ mới để phân tích dữ liệu & & dự đoán các thuộc tính mới từ dữ liệu đó. Điều này trái ngược với dữ liệu dạng bảng, e.g.: bảng cơ sở dữ liệu, khung dữ liệu hoặc bảng tính, trong đó dữ liệu được cố định theo hàng & cột.

To describe & learn from edges between nodes, we need a way to write them down. This can be explicitly, quickly, can see describing things in this way becomes unwieldy & might be repeating redundant information. Luckily, there are many mathematical formalisms for describing relations in graphs. 1 of most common: describe *adjacency matrix*. Notice: adjacency matrix is symmetric across diagonal & all values are 1s or 0s. Adjacency matrix of a graph is an important concept that makes it easy to observe all connections of a graph in a single table. Here assumed: there is no directionability in our graphs, i.e., if 0 is connected to 1, then 1 is also connected to 0. This is known as an *undirected graph*. Undirected graphs can be easily inferred from an adjacency matrix because, in this case, matrix is symmetric across diagonal, upper-right triangle is reflected onto bottom-left.

– Để mô tả & học từ các cạnh giữa các nút, chúng ta cần 1 cách để viết chúng ra. Điều này có thể rõ ràng, nhanh chóng, có thể thấy việc mô tả mọi thứ theo cách này trở nên cồng kềnh & có thể lặp lại thông tin dư thừa. May mắn thay, có nhiều công thức toán học để mô tả các mối quan hệ trong đồ thị. 1 trong những công thức phổ biến nhất: mô tả *adjacency matrix*. Lưu ý: ma trận kề là đối xứng qua đường chéo & tất cả các giá trị là 1 hoặc 0. Ma trận kề của đồ thị là 1 khái niệm quan trọng giúp dễ dàng quan sát tất cả các kết nối của đồ thị trong 1 bảng duy nhất. Ở đây giả sử: không có tính định hướng trong đồ thị của chúng ta, i.e., nếu 0 được kết nối với 1, thì 1 cũng được kết nối với 0. Đây được gọi là *undirected graph*. Đồ thị vô hướng có thể dễ dàng suy ra từ ma trận kề vì, trong trường hợp này, ma trận đối xứng qua đường chéo, tam giác trên cùng bên phải được phản chiếu xuống dưới cùng bên trái.

Also assume: all relations between nodes are identical. If we wanted relation of nodes B-E to mean more than relation of nodes B-A, then we could increase weight of this edge. This translates to increasing value in adjacency matrix, making entry for B-A edge equal to 10 instead of 1, e.g.

– Cũng giả sử: tất cả các mối quan hệ giữa các nút đều giống hệt nhau. Nếu chúng ta muốn mối quan hệ giữa các nút B-E có ý nghĩa hơn mối quan hệ giữa các nút B-A, thì chúng ta có thể tăng trọng số của cạnh này. Điều này tương đương với việc tăng giá trị trong ma trận kề, e.g., nhập giá trị cho cạnh B-A bằng 10 thay vì 1.

Graphs where all relations are of equal importance are known as *unweighted graphs* & can also be easily observed from adjacency matrix because all graph entries are either 1s or 0s. Graphs where edges have multiple values are known as *weighted*.

– Đồ thị mà tất cả các mối quan hệ đều có tầm quan trọng như nhau được gọi là *unweighted graphs* & cũng có thể dễ dàng quan sát được từ ma trận kề vì tất cả các mục đồ thị đều là 1 hoặc 0. Đồ thị mà các cạnh có nhiều giá trị được gọi là *weighted*.

If any of nodes in graph do not have an edge that connects to itself, then nodes will also have 0s at their own value in adjacency matrix (0s along diagonal), i.e., a graph does not have self-loops. A *self-loop* occurs when a node has an edge that connects to that same node. To add a self-loop, we just make value for that node nonzero at its position in diagonal.

– Nếu bất kỳ nút nào trong đồ thị không có cạnh nối với chính nó, thì các nút cũng sẽ có giá trị 0 tại chính nó trong ma trận kề (các giá trị 0 dọc theo đường chéo), i.e., đồ thị không có vòng lặp tự thân. 1 vòng lặp tự thân xảy ra khi 1 nút có cạnh nối với chính nút đó. Để thêm 1 vòng lặp tự thân, chúng ta chỉ cần gán giá trị khác không cho nút đó tại vị trí của nó trên đường chéo.

In practice, an adjacency matrix is only 1 of many ways to describe relations in a graph. Others include adjacency lists, edge lists, or an incidence matrix. Understanding these types of data structures well is vital to graph-based learning. If you are unfamiliar with these terms, or need a refresher, recommend looking through appendix A, which has additional details & explanations.

– Trên thực tế, ma trận kề chỉ là 1 trong nhiều cách để mô tả các mối quan hệ trong đồ thị. Các cách khác bao gồm danh sách kề, danh sách cạnh, hoặc ma trận liên quan. Việc hiểu rõ các loại cấu trúc dữ liệu này rất quan trọng đối với việc học tập dựa trên đồ thị. Nếu bạn chưa quen với các thuật ngữ này hoặc cần ôn tập lại, xem Phụ lục A, trong đó có thêm chi tiết & giải thích.

∗ 1.2.2. Different types of graphs. Understanding many different types of graphs can help us work out what methods to use to analyze & transform graph, & what ML methods to apply. In following, give a very quick overview of some of most common properties for graphs to have.

– Hiểu biết về nhiều loại đồ thị khác nhau có thể giúp chúng ta tìm ra phương pháp nào cần sử dụng để phân tích & biến đổi đồ thị, & áp dụng phương pháp ML nào. Sau đây, chúng tôi sẽ giới thiệu sơ lược về 1 số thuộc tính phổ biến nhất của đồ thị.

· Homogeneous & Heterogeneous Graphs. Most basic graphs are *homogeneous graphs*, which are made up of 1 type of node & 1 type of edge. Consider a homogeneous graph that describes a recruitment network. In this type of graph, nodes would represent job candidates, & edges would represent relationships between candidates.

– Đồ thị Đồng nhất & Đồ thị Không Đồng nhất. Hầu hết các đồ thị cơ bản là *đồ thị đồng nhất*, được tạo thành từ 1 loại nút & 1 loại cạnh. Xem xét 1 đồ thị đồng nhất mô tả 1 mạng lưới tuyển dụng. Trong loại đồ thị này, các nút sẽ đại diện cho các ứng viên xin việc, & cạnh sẽ đại diện cho mối quan hệ giữa các ứng viên.

If want to expand power of our graph to describe our recruitment network, could give it more types of nodes & edges, making it a *heterogeneous graphs*. With this expansion, some nodes may be candidates & others may be companies. Edges could now consist of relationships between candidates & current or past employment of job candidates at companies. See Fig. 1.2: A homogeneous graph & a heterogeneous graph. Here, shade of a node or edge represents its type or class. For homogeneous graph, all nodes are of same type, & all edges are of same type. For heterogeneous graph, nodes & edges have multiple types for a comparison of a homogeneous graph (all nodes or edges have same shade) with a heterogeneous graph (nodes & edges have a variety of shades).

– Nếu muốn mở rộng sức mạnh của đồ thị để mô tả mạng lưới tuyển dụng, có thể cung cấp cho nó nhiều loại nút &

cạnh hơn, biến nó thành *đồ thị không đồng nhất*. Với sự mở rộng này, 1 số nút có thể là ứng viên & những nút khác có thể là công ty. Các cạnh bây giờ có thể bao gồm các mối quan hệ giữa ứng viên & việc làm hiện tại hoặc trước đây của ứng viên tại các công ty. Xem Hình 1.2: Đồ thị đồng nhất & đồ thị không đồng nhất. Ở đây, sắc thái của 1 nút hoặc cạnh biểu thị loại hoặc lớp của nó. Đối với đồ thị đồng nhất, tất cả các nút đều cùng loại, & tất cả các cạnh đều cùng loại. Đối với đồ thị không đồng nhất, các nút & cạnh có nhiều loại để so sánh đồ thị đồng nhất (tất cả các nút hoặc cạnh có cùng sắc thái) với đồ thị không đồng nhất (các nút & cạnh có nhiều sắc thái khác nhau).

- Bipartite graphs. Similar to heterogeneous graphs, *bipartite graphs* also can be separated or partitioned into different subsets. However, bipartite graphs (Fig. 1.3: A bipartite graph. There are 2 types of nodes (2 shades of circles). In a bipartite graph, nodes cannot be connected to nodes of same type. This is also an example of a heterogeneous graph.) have a very specific network structure s.t. nodes in each subset connect to nodes outside of their subset & not inside. Later, discuss recommendation system & Pinterest graph. This graph is bipartite because 1 set of nodes (pins) connects another set of nodes (boards) but not to nodes within their set (pins).
  – Đồ thị 2 phần. Tương tự như đồ thị không đồng nhất, *Đồ thị 2 phần* cũng có thể được tách hoặc phân vùng thành các tập con khác nhau. Tuy nhiên, đồ thị 2 phần (Hình 1.3: Đồ thị 2 phần. Có 2 loại nút (2 sắc thái của hình tròn). Trong đồ thị 2 phần, các nút không thể được kết nối với các nút cùng loại. Đây cũng là 1 ví dụ về đồ thị không đồng nhất.) có cấu trúc mạng rất cụ thể, e.g.: các nút trong mỗi tập con kết nối với các nút bên ngoài tập con của chúng & không kết nối với các nút bên trong. Sau đó, thảo luận về hệ thống đề xuất & đồ thị Pinterest. Đồ thị này là 2 phần vì 1 tập hợp các nút (ghim) kết nối với 1 tập hợp các nút khác (bảng) nhưng không kết nối với các nút trong tập hợp của chúng (ghim).
- Cyclic graphs, acyclic graphs, & directed acyclic graphs. A graph is *cyclic* if it allows you to start at a node, travel along its edge, & return to starting node without retracing any steps, creating a circular path within graph. In contrast, in an *acyclic* graph, no matter which path you take from any starting node, you cannot return to starting point without backtracking. These graphs, as shown in Fig. 1.4: A cyclic graph, an acyclic graph, & a DAG. In cyclic graph, cycle is shown by arrows (directed edges) connecting nodes A-E-D-C-B-A. Note 2 nodes, F & G are part of graph, but not part of its defining cycle. Acyclic graph is composed of undirected edges, & no cycle is possible. In DAG, all directed edges flow in 1 direction, from A to F., often resemble tree-like structures or paths that do not loop back on themselves.
  – Đồ thị tuần hoàn, đồ thị phi chu trình, & đồ thị phi chu trình có hướng. 1 đồ thị là *tuần hoàn* nếu nó cho phép bạn bắt đầu tại 1 nút, di chuyển dọc theo cạnh của nó, & quay lại nút bắt đầu mà không cần quay lại bất kỳ bước nào, tạo ra 1 đường tròn trong đồ thị. Ngược lại, trong đồ thị *phi chu trình*, bất kể bạn đi theo đường nào từ bất kỳ nút bắt đầu nào, bạn không thể quay lại điểm bắt đầu mà không quay lại. Các đồ thị này, như được thể hiện trong Hình 1.4: Đồ thị tuần hoàn, đồ thị phi chu trình, & DAG. Trong đồ thị tuần hoàn, chu trình được biểu diễn bằng các mũi tên (cạnh có hướng) nối các nút A-E-D-C-B-A. Lưu ý 2 nút, F & G là 1 phần của đồ thị, nhưng không phải là 1 phần của chu trình xác định của nó. Đồ thị phi chu trình bao gồm các cạnh không có hướng, & không thể có chu trình. Trong DAG, tất cả các cạnh có hướng đều chảy theo 1 hướng, từ A đến F., thường giống các cấu trúc giống cây hoặc các đường dẫn không vòng lại với chính chúng.
  While both cyclic & acyclic graphs can be either undirected or directed, a *directed acyclic graph (DAG)* is a specific type of acyclic graph that is exclusively directed. In a DAG, all edges have a direction, & no cycles are allowed. DAGs represent 1-way relationships where we can't follow arrows & end up back at starting point. This characteristic makes DAGs essential in causal analysis, as they reflect causal structures where causality is assumed to be undirectional. E.g., A can cause B, but B can't simultaneously cause A. This unidirectional nature aligns perfectly with structure of DAGs, making them ideal for modeling workflow processes, dependency chains, & causal relationships in various fields.
  – Trong khi cả đồ thị tuần hoàn & đồ thị phi tuần hoàn đều có thể vô hướng hoặc có hướng, *đồ thị phi tuần hoàn có hướng (DAG)* là 1 loại đồ thị phi tuần hoàn cụ thể chỉ có hướng. Trong DAG, tất cả các cạnh đều có hướng, & không cho phép chu trình. DAG biểu diễn các mối quan hệ 1 chiều, trong đó chúng ta không thể theo mũi tên & kết thúc ở điểm xuất phát. Đặc điểm này làm cho DAG trở nên thiết yếu trong phân tích nhân quả, vì chúng phản ánh các cấu trúc nhân quả trong đó quan hệ nhân quả được coi là vô hướng. Ví dụ: A có thể gây ra B, nhưng B không thể đồng thời gây ra A. Bản chất đơn hướng này hoàn toàn phù hợp với cấu trúc của DAG, khiến chúng trở nên lý tưởng để mô hình hóa các quy trình công việc, chuỗi phụ thuộc, & các mối quan hệ nhân quả trong nhiều lĩnh vực khác nhau.
- Knowledge graphs. A *knowledge graph* is a specialized type of heterogeneous graph that represents data with enriched semantic meaning, capturing not only relationships between different entities but also context & nature of these relationships. Unlike conventional graphs, which primarily emphasize structure & connectivity, a knowledge graph incorporates metadata & follow specific schemas to provide deeper contextual information. This allows for advanced reasoning & querying capabilities, e.g. identifying patterns, uncovering specific types of connections, or inferring new relationships.
  – Đồ thị tri thức. Đồ thị tri thức là 1 loại đồ thị không đồng nhất chuyên biệt, biểu diễn dữ liệu với ý nghĩa ngữ nghĩa phong phú, không chỉ nắm bắt mối quan hệ giữa các thực thể khác nhau mà còn cả bối cảnh & bản chất của các mối quan hệ này. Không giống như đồ thị thông thường, chủ yếu nhấn mạnh vào cấu trúc & kết nối, đồ thị tri thức kết hợp siêu dữ liệu & tuân theo các lược đồ cụ thể để cung cấp thông tin ngữ cảnh sâu hơn. Điều này cho phép khả năng suy luận & truy vấn nâng cao, e.g.: xác định các mẫu, khám phá các loại kết nối cụ thể hoặc suy ra các mối quan hệ mới.
  In example of an academic research network at a university, a knowledge graph might represent various entities e.g. Professors, Students, Papers, & Research Topics, & explicitly define relationships between them. E.g., Professors & Students could be associated with Papers through an Authorship relationship, while Professors might also Supervise Students. Furthermore, graph would reflect hierarchical structures, e.g., Professors & Students being categorized under Departments. Can see this knowledge graph depicted in Fig. 1.5: A knowledge graph representing an academic research network within a university's physics department. Graph illustrates both hierarchical relationships, e.g., professors & students

as members of department, & behavioral relationships, e.g., professors supervising students & authoring papers. Entities e.g. Professors, Students, Papers, & Topics are connected through semantically meaningful relationships (Supervises, Wrote, Inspires). Entities also have detailed features (Name, Department, Type) providing further context. Semantic connections & features enable advanced querying & analysis of complex academic interactions.

– Trong ví dụ về mạng lưới nghiên cứu học thuật tại 1 trường đại học, biểu đồ kiến thức có thể biểu diễn nhiều thực thể khác nhau, e.g.: Giáo sư, Sinh viên, Bài báo, & Chủ đề nghiên cứu, & định nghĩa rõ ràng mối quan hệ giữa chúng. Ví dụ: Giáo sư & Sinh viên có thể được liên kết với Bài báo thông qua mối quan hệ Tác giả, trong khi Giáo sư cũng có thể Giám sát Sinh viên. Hơn nữa, biểu đồ sẽ phản ánh các cấu trúc phân cấp, e.g.: Giáo sư & Sinh viên được phân loại theo Khoa. Có thể xem biểu đồ kiến thức này được mô tả trong Hình 1.5: Biểu đồ kiến thức biểu diễn mạng lưới nghiên cứu học thuật trong khoa vật lý của 1 trường đại học. Biểu đồ minh họa cả các mối quan hệ phân cấp, e.g.: giáo sư & sinh viên là thành viên của khoa, & các mối quan hệ hành vi, e.g.: giáo sư hướng dẫn sinh viên & biên soạn bài báo. Các thực thể, e.g.: Giáo sư, Sinh viên, Bài báo, & Chủ đề được kết nối thông qua các mối quan hệ có ý nghĩa ngữ nghĩa (Giám sát, Viết, Truyền cảm hứng). Các thực thể cũng có các tính năng chi tiết (Tên, Khoa, Loại) cung cấp thêm ngữ cảnh. Kết nối ngữ nghĩa & các tính năng cho phép truy vấn nâng cao & phân tích các tương tác học thuật phức tạp.

Note 1. *Should use multigraphs to further represent knowledge graphs.*

A key feature of knowledge graphs is their ability to provide explicit context. Unlike conventional heterogeneous graphs, which display different types of entities & their basic connections without detailed semantic meaning, knowledge graphs go further by defining specific types & meanings of relationships. E.g., while a traditional graph might show that Professors are connected to Departments or that Students are linked to Papers, a knowledge graph would specify that Professors supervise Students or that Students & Professors Wrote Papers. This added layer of meaning enables more powerful querying & analysis, making knowledge graphs particularly valuable in fields e.g. NLP, recommendation systems, & academic research analysis.

– 1 đặc điểm quan trọng của đồ thị tri thức là khả năng cung cấp ngữ cảnh rõ ràng. Không giống như các đồ thị không đồng nhất thông thường, vốn hiển thị các loại thực thể khác nhau & các kết nối cơ bản của chúng mà không có ý nghĩa ngữ nghĩa chi tiết, đồ thị tri thức tiến xa hơn bằng cách xác định các loại & ý nghĩa cụ thể của các mối quan hệ. Ví dụ: trong khi đồ thị truyền thống có thể hiển thị Giáo sư được kết nối với Khoa hoặc Sinh viên được liên kết với Bài báo, đồ thị tri thức sẽ chỉ rõ Giáo sư hướng dẫn Sinh viên hoặc Sinh viên & Giáo sư viết Bài báo. Lớp ý nghĩa bổ sung này cho phép truy vấn & phân tích mạnh mẽ hơn, khiến đồ thị tri thức đặc biệt có giá trị trong các lĩnh vực như NLP, hệ thống đề xuất, & phân tích nghiên cứu học thuật.

· Hypergraphs. 1 of more complex & difficult graphs to work with is hypergraph. *Hypergraphs* are those where a single edge can be connected to multiple different nodes. For graphs that are not hypergraphs, edges are used to connected exactly 2 nodes (or a node to itself for self-loops). As shown in Fig. 1.6: 1 undirected hypergraph, illustrated in 2 ways. On left, have a graph whose edges are represented by shaded areas, marked by letters, & whose vertices are dots, marked by numbers. On right, have a graph whose edge lines (marked by letters) connect up to 3 nodes (circles marked by numbers)., edges in a hypergraph can connect between any number of nodes. Complexity of a hypergraph is reflected in its adjacency data. For typical graphs, network connectivity is represented by a 2D adjacency matrix. For hypergraphs, adjacency matrix extends to a higher dimensional tensor, referred to as an *incidence tensor*. This tensor is N-dimensional, where N is maximum number of nodes connected by a single edge. An example of a hypergraph might be a communication platform that allows for group chats as well as single person conversations. In an ordinary graph, edges would only connect 2 people. In a hypergraph, 1 hyperedge could connect multiple people, representing a group chat.

– Siêu đồ thị. 1 trong những đồ thị phức tạp hơn & khó làm việc hơn là siêu đồ thị. Siêu đồ thị là những đồ thị mà 1 cạnh đơn có thể được kết nối với nhiều nút khác nhau. Đối với những đồ thị không phải là siêu đồ thị, các cạnh được sử dụng để kết nối chính xác 2 nút (hoặc 1 nút với chính nó đối với các vòng lặp tự thân). Như thể hiện trong Hình 1.6: 1 siêu đồ thị vô hướng, được minh họa theo 2 cách. Bên trái, có 1 đồ thị mà các cạnh được biểu diễn bằng các vùng tô bóng, được đánh dấu bằng các chữ cái, & có các đỉnh là các dấu chấm, được đánh dấu bằng các số. Bên phải, có 1 đồ thị mà các đường cạnh (được đánh dấu bằng các chữ cái) kết nối tối đa 3 nút (các vòng tròn được đánh dấu bằng các số)., các cạnh trong siêu đồ thị có thể kết nối giữa bất kỳ số lượng nút nào. Độ phức tạp của siêu đồ thị được phản ánh trong dữ liệu kề của nó. Đối với các đồ thị thông thường, kết nối mạng được biểu diễn bằng ma trận kề 2D. Đối với siêu đồ thị, ma trận kề mở rộng đến 1 tenxơ có chiều cao hơn, được gọi là tenxơ *incidence tenxơ*. Tenxơ này là tenxơ N chiều, trong đó N là số lượng nút tối đa được kết nối bởi 1 cạnh duy nhất. 1 ví dụ về siêu đồ thị có thể là 1 nền tảng giao tiếp cho phép trò chuyện nhóm cũng như trò chuyện cá nhân. Trong 1 đồ thị thông thường, các cạnh chỉ kết nối 2 người. Trong 1 siêu đồ thị, 1 siêu cạnh có thể kết nối nhiều người, đại diện cho 1 cuộc trò chuyện nhóm.

∗ 1.2.3. Graph-based learning. Graphs are ubiquitous in our everyday life. *Graph-based learning* takes graphs as input data to build models that give insight into questions about this data. Later in this chap, look at different examples of graph data as well as at sort of questions & tasks we can use graph-based learning to answer.

– Học tập dựa trên đồ thị. Đồ thị hiện diện khắp nơi trong cuộc sống hàng ngày của chúng ta. *Học tập dựa trên đồ thị* sử dụng đồ thị làm dữ liệu đầu vào để xây dựng các mô hình cung cấp thông tin chi tiết về các câu hỏi liên quan đến dữ liệu này. Ở phần sau của chương này, xem xét các ví dụ khác nhau về dữ liệu đồ thị cũng như các loại câu hỏi & bài tập mà chúng ta có thể sử dụng học tập dựa trên đồ thị để trả lời.

Graph-based learning uses a variety of ML methods to build *representations* of graphs. These representations are then used for downstream tasks e.g. node or link prediction or graph classification. In Chap. 2, learn about 1 of essential tools in graph-based learning, building embeddings. Briefly, embeddings are *low-dimensional* vector representations. Can build an embedding of different nodes, edges, or entire graphs, & there are a number of different ways to do this e.g. Node2Vec

(N2V) or DeepWalk algorithms.
– Học tập dựa trên đồ thị sử dụng nhiều phương pháp học máy khác nhau để xây dựng *biểu diễn* của đồ thị. Các biểu diễn này sau đó được sử dụng cho các tác vụ hạ nguồn, e.g. dự đoán nút hoặc liên kết hoặc phân loại đồ thị. Trong Chương 2, tìm hiểu về 1 trong những công cụ thiết yếu trong học tập dựa trên đồ thị, đó là xây dựng nhúng. Nói 1 cách ngắn gọn, nhúng là biểu diễn vectơ *chiều thấp*. Có thể xây dựng nhúng của các nút, cạnh hoặc toàn bộ đồ thị khác nhau, & có 1 số cách khác nhau để thực hiện việc này, e.g. thuật toán Node2Vec (N2V) hoặc DeepWalk.
Methods for analysis on graph data have been around for a long time, at least as early as 1950s when *clique methods* used certain features of a graph to identify subsets or communities in graph data [4].
– Các phương pháp phân tích dữ liệu đồ thị đã có từ rất lâu, ít nhất là từ những năm 1950 khi *clique methods* sử dụng 1 số tính năng nhất định của đồ thị để xác định các tập hợp con hoặc cộng đồng trong dữ liệu đồ thị.
1 of most famous graph-based algorithms is PageRank, which was developed by Larry Page & Sergey Brin in 1996 & formed basis for Google's search algorithms. Some believe: this algorithm was a key element in company's meteoric rise in following years. This highlights that a successful graph-based learning algorithm can have a huge effect.
– 1 trong những thuật toán dựa trên đồ thị nổi tiếng nhất là PageRank, được phát triển bởi Larry Page & Sergey Brin vào năm 1996 & tạo nền tảng cho các thuật toán tìm kiếm của Google. 1 số người tin rằng: thuật toán này là yếu tố then chốt cho sự phát triển vượt bậc của công ty trong những năm tiếp theo. Điều này nhấn mạnh rằng 1 thuật toán học dựa trên đồ thị thành công có thể mang lại hiệu quả to lớn.
These methods are only a small subset of graph-based learning & analysis techniques. Others include belief propagation [5], graph kernel methods [6], label propagation [7], & isomaps [8]. However, in this book, focus on 1 of newest & most exciting additions to family of graph-based learning techniques: GNNs.
– Những phương pháp này chỉ là 1 tập hợp con nhỏ của các kỹ thuật học tập dựa trên đồ thị & phân tích. Các phương pháp khác bao gồm truyền bá niềm tin [5], phương pháp hạt nhân đồ thị [6], truyền bá nhãn [7], & isomaps [8]. Tuy nhiên, trong cuốn sách này, chúng tôi tập trung vào 1 trong những bổ sung mới nhất & thú vị nhất cho nhóm kỹ thuật học tập dựa trên đồ thị: GNN.

∗ 1.2.4. What is a GNN? GNNs combine graph-based learning with DL, i.e., neural networks are used to build embeddings & process relational data. An overview of inner workings of a GNN is shown in Fig. 1.7: An overview of how GNNs work. An input graph is passed to a GNN. GNN then uses neural networks to transform graph features e.g. nodes or edges into nonlinear embeddings through a process known as message passing. These embeddings are then tuned to specific unknown properties using training data. After GNN is trained, it can predict unknown features of a graph.
– GNN kết hợp học tập dựa trên đồ thị với DL, i.e., mạng nơ-ron được sử dụng để xây dựng các nhúng & xử lý dữ liệu quan hệ. Tổng quan về hoạt động bên trong của GNN được thể hiện trong Hình 1.7: Tổng quan về cách thức hoạt động của GNN. 1 đồ thị đầu vào được truyền đến GNN. Sau đó, GNN sử dụng mạng nơ-ron để chuyển đổi các đặc trưng đồ thị, e.g. các nút hoặc cạnh, thành các nhúng phi tuyến tính thông qua 1 quá trình được gọi là truyền thông điệp. Các nhúng này sau đó được điều chỉnh theo các thuộc tính cụ thể chưa biết bằng cách sử dụng dữ liệu huấn luyện. Sau khi GNN được huấn luyện, nó có thể dự đoán các đặc trưng chưa biết của đồ thị.
GNNs allows you to represent & learn from graphs, including their constituent nodes, edges, & features. In particular, many methods of GNNs are built specifically to scale effectively with size & complexity of a graph, i.e., GNNs can operate on huge graphs. In this sense, GNNs provide analogous advantages to relational data as convolutional neural networks have given for image-based data & computer vision.
– GNN cho phép bạn biểu diễn & học từ các đồ thị, bao gồm các nút, cạnh, & đặc trưng cấu thành của chúng. Đặc biệt, nhiều phương pháp GNN được xây dựng chuyên biệt để mở rộng hiệu quả theo kích thước & độ phức tạp của đồ thị, i.e., GNN có thể hoạt động trên các đồ thị rất lớn. Theo nghĩa này, GNN mang lại những lợi thế tương tự cho dữ liệu quan hệ như mạng nơ-ron tích chập đã mang lại cho dữ liệu dựa trên hình ảnh & thị giác máy tính.
Historically, applying traditional ML methods to graph data structures has been challenging because graph data, when represented in grid-like formats & data structures, can lead to massive repetitions of data. To address this, graph-based learning focuses on approaches that are *permutation invariant*, i.e., ML method is uninfluenced by ordering of graph representation. In concrete terms, it means: we can shuffle rows & columns of adjacency matrix without affecting our algorithm's performance. Whenever we are working with data that contains relational data, i.e., has an adjacency matrix, then we want to use a ML method that is permutation invariant to make our method more general & efficient. Although GNNs can be applied to all graph data, GNNs are especially useful because they can deal with huge graph datasets & typically perform better than other ML methods.
– Theo truyền thống, việc áp dụng các phương pháp ML truyền thống vào cấu trúc dữ liệu đồ thị là 1 thách thức vì dữ liệu đồ thị, khi được biểu diễn ở định dạng dạng lưới & cấu trúc dữ liệu, có thể dẫn đến sự lặp lại dữ liệu rất lớn. Để giải quyết vấn đề này, học dựa trên đồ thị tập trung vào các phương pháp *permutation invariant*, i.e., phương pháp ML không bị ảnh hưởng bởi thứ tự biểu diễn đồ thị. Nói 1 cách cụ thể, điều này có nghĩa là: chúng ta có thể xáo trộn các hàng & cột của ma trận kề mà không ảnh hưởng đến hiệu suất của thuật toán. Bất cứ khi nào chúng ta làm việc với dữ liệu có chứa dữ liệu quan hệ, i.e., có ma trận kề, thì chúng ta muốn sử dụng phương pháp ML không thay đổi hoán vị để làm cho phương pháp của chúng ta tổng quát hơn & hiệu quả hơn. Mặc dù GNN có thể được áp dụng cho tất cả dữ liệu đồ thị, nhưng GNN đặc biệt hữu ích vì chúng có thể xử lý các tập dữ liệu đồ thị khổng lồ & thường hoạt động tốt hơn các phương pháp ML khác.
Permutation invariances are a type of *inductive bias*, or an algorithm's learning bias, & are powerful tools for designing ML algorithms [1]. Need for permutation-invariant approaches is 1 of central reasons that graph-based learning has increased in popularity in recent years.

– Bất biến hoán vị là 1 loại *thiên vị quy nạp*, hay thiên vị học tập của thuật toán, & là những công cụ mạnh mẽ để thiết kế các thuật toán ML [1]. Nhu cầu về các phương pháp bất biến hoán vị là 1 trong những lý do chính khiến học tập dựa trên đồ thị ngày càng phổ biến trong những năm gần đây.

Insights. Being designed for permutation-invariant data comes with some drawbacks along with its advantages. GNNs are not as well suited for other data, e.g. images or tables. While this might seem obvious, images & tables are not permutation invariant & therefore not a good fit for GNNs. If we shuffle rows & columns of an image, then we scramble input. Instead, ML algorithms for images seek *translational invariance*, i.e., we can translate (shift) object in an image, & it won't affect performance of algorithm. Other neural networks, e.g. convolutional neural networks (CNNs) typically perform much better on images.

– Việc được thiết kế cho dữ liệu bất biến hoán vị đi kèm với 1 số nhược điểm bên cạnh những ưu điểm của nó.

GNN không phù hợp lắm với các dữ liệu khác, e.g. hình ảnh hoặc bảng. Mặc dù điều này có vẻ hiển nhiên, nhưng hình ảnh & bảng không bất biến hoán vị & do đó không phù hợp với GNN. Nếu chúng ta xáo trộn các hàng & cột của 1 hình ảnh, thì chúng ta sẽ xáo trộn dữ liệu đầu vào. Thay vào đó, các thuật toán ML cho hình ảnh tìm kiếm *translational invariance*, i.e., chúng ta có thể dịch chuyển (shift) đối tượng trong 1 hình ảnh, & điều này sẽ không ảnh hưởng đến hiệu suất của thuật toán. Các mạng nơ-ron khác, e.g. mạng nơ-ron tích chập (CNN) thường hoạt động tốt hơn nhiều trên hình ảnh.

∗ 1.2.5. Differences between tabular & graph data. Graph data includes all data with some relational content, making it a powerful way to represent complex connections. While graph data might initially seem distinct from traditional tabular data, many datasets that are typically represented in tables can be recreated as graphs with some data engineering & imagination. Take a closer look at Titanic dataset, a classic example in ML, & explore how it can be transformed from a table format to a graph format.

– Sự khác biệt giữa dữ liệu dạng bảng & dữ liệu đồ thị. Dữ liệu đồ thị bao gồm tất cả dữ liệu có nội dung quan hệ, khiến nó trở thành 1 phương pháp mạnh mẽ để biểu diễn các kết nối phức tạp. Mặc dù ban đầu dữ liệu đồ thị có vẻ khác biệt so với dữ liệu dạng bảng truyền thống, nhưng nhiều tập dữ liệu thường được biểu diễn dưới dạng bảng có thể được tái tạo dưới dạng đồ thị với 1 chút kỹ thuật dữ liệu & trí tưởng tượng. Hãy xem xét kỹ hơn tập dữ liệu Titanic, 1 ví dụ kinh điển trong ML, & khám phá cách nó có thể được chuyển đổi từ định dạng bảng sang định dạng đồ thị.

Titanic dataset describes passengers on Titanic, a ship that famously met an untimely end when it collided with an iceberg. Historically, this datasets has been analyzed in tabular format, containing rows for each passenger with columns representing features e.g. age, gender, fare, class, & survival status. However, dataset also contains rich, unexplored relationships that are not immediately visible in a table format Fig. 1.8: Titanic Dataset is usually displayed & analyzed using a table format.

– Bộ dữ liệu Titanic mô tả hành khách trên Titanic, 1 con tàu nổi tiếng đã gặp phải cái chết bất ngờ khi va chạm với 1 tảng băng trôi. Trước đây, bộ dữ liệu này được phân tích theo định dạng bảng, bao gồm các hàng cho mỗi hành khách & các cột biểu diễn các đặc điểm như tuổi, giới tính, giá vé, hạng ghế, & tình trạng sống sót. Tuy nhiên, bộ dữ liệu cũng chứa các mối quan hệ phong phú, chưa được khám phá mà không thể hiển thị ngay lập tức ở định dạng bảng Hình 1.8: Bộ dữ liệu Titanic thường được hiển thị & phân tích bằng định dạng bảng.

· Recasting Titanic dataset as a graph. To transform Titanic dataset into a graph, need to consider how to represent underlying relationships between passengers as nodes & edges:

1. Nodes: In graph, each passenger can be represented as a node. Can also introduce nodes for other entities, e.g. cabins, families, or even groups e.g. "3rd-class passengers".
2. Edges represent relationships or connections between these nodes, e.g.: Passengers who are family members (siblings, spouses, parents, or children) based on available data; Passengers who share a cabin or were traveling together; Social or business relationships that might be inferred from shared ticket numbers, last names, or other identifying features.

– Tái cấu trúc tập dữ liệu Titanic dưới dạng đồ thị. Để chuyển đổi tập dữ liệu Titanic thành đồ thị, cần xem xét cách biểu diễn các mối quan hệ cơ bản giữa hành khách dưới dạng các nút & cạnh:

1. Nút: Trong đồ thị, mỗi hành khách có thể được biểu diễn dưới dạng 1 nút. Cũng có thể giới thiệu các nút cho các thực thể khác, e.g.: cabin, gia đình hoặc thậm chí các nhóm, e.g.: "hành khách hạng 3".
2. Cạnh biểu diễn các mối quan hệ hoặc kết nối giữa các nút này, e.g.: Hành khách là thành viên gia đình (anh chị em ruột, vợ/chồng, cha mẹ hoặc con cái) dựa trên dữ liệu có sẵn; Hành khách ở chung cabin hoặc đi cùng nhau; Các mối quan hệ xã hội hoặc kinh doanh có thể được suy ra từ số vé, họ hoặc các đặc điểm nhận dạng chung khác.

To construct this graph, need to use existing information in table & potentially enrich it with secondary data sources or assumptions (e.g., linking last names to create family groups). This process converts tabular data into a graph-based structure, shown in Fig. 1.9: Titanic dataset, showing family relationships of people on Titanic visualized as a graph. Here, can see that there was a rich social network as well as many passengers with unknown family ties., where each edge & node encapsulates meaningful relational data.

– Để xây dựng biểu đồ này, cần sử dụng thông tin hiện có trong bảng & có thể làm giàu nó bằng các nguồn dữ liệu thứ cấp hoặc giả định (e.g.: liên kết họ để tạo nhóm gia đình). Quá trình này chuyển đổi dữ liệu dạng bảng thành cấu trúc dạng biểu đồ, được hiển thị trong Hình 1.9: Bộ dữ liệu Titanic, thể hiện mối quan hệ gia đình của những người trên Titanic được trực quan hóa dưới dạng biểu đồ. Ở đây, có thể thấy rằng có 1 mạng lưới xã hội phong phú cũng như nhiều hành khách có mối quan hệ gia đình chưa rõ ràng., trong đó mỗi cạnh & nút đóng gói dữ liệu quan hệ có ý nghĩa.

· How graph data adds depth & meaning. Once dataset is represented as a graph, it provides a much deeper view of social & familial connections between passengers, e.g.,:

1. *Family relationships*: Graph clearly shows how certain passengers were related (e.g., as parents, children, or siblings). This could help us understand survival patterns, as family members might have behaved differently in a crisis than individuals traveling alone.
2. *Social networks*: Beyond families, graph could reveal broader social networks (e.g., friendships or business connections), which could be important factors in analyzing behavior & outcomes.
3. *Community insights*: Graph structure also allows for community detection algorithms to identify clusters of related or connected passengers, which may reveal new insights into survival rates, rescue patterns, or other behaviors.

Graph representations add depth by specifying connections that might not be obvious in a tabular format. E.g., understanding who traveled together, who shared a cabin, or who had social or family ties can provide more context on survival rates & passenger behavior. This is crucial for tasks e.g. node prediction, where we want to predict attributes or outcomes based on relationships represented in graph.

– Cách dữ liệu đồ thị tăng thêm chiều sâu & ý nghĩa. Khi tập dữ liệu được biểu diễn dưới dạng đồ thị, nó cung cấp cái nhìn sâu sắc hơn nhiều về các mối quan hệ xã hội & gia đình giữa các hành khách, e.g.:

1. *Mối quan hệ gia đình*: Đồ thị cho thấy rõ mối quan hệ của 1 số hành khách nhất định (e.g.: cha mẹ, con cái hoặc anh chị em ruột). Điều này có thể giúp chúng ta hiểu được các mô hình sinh tồn, vì các thành viên trong gia đình có thể đã hành xử khác nhau trong khủng hoảng so với những cá nhân đi du lịch 1 mình.
2. *Mạng xã hội*: Ngoài gia đình, đồ thị có thể tiết lộ các mạng xã hội rộng hơn (e.g.: tình bạn hoặc kết nối kinh doanh), đây có thể là những yếu tố quan trọng trong việc phân tích hành vi & kết quả.
3. *Thông tin chi tiết về cộng đồng*: Cấu trúc đồ thị cũng cho phép các thuật toán phát hiện cộng đồng xác định các cụm hành khách có liên quan hoặc kết nối, điều này có thể tiết lộ những hiểu biết mới về tỷ lệ sống sót, mô hình cứu hộ hoặc các hành vi khác.

Biểu diễn đồ thị tăng thêm chiều sâu bằng cách chỉ định các kết nối có thể không rõ ràng ở định dạng bảng. E.g., việc hiểu rõ ai đi cùng nhau, ai ở chung cabin, hoặc ai có mối quan hệ xã hội hoặc gia đình có thể cung cấp thêm bối cảnh về tỷ lệ sống sót & hành vi của hành khách. Điều này rất quan trọng đối với các tác vụ như dự đoán nút, trong đó chúng ta muốn dự đoán các thuộc tính hoặc kết quả dựa trên các mối quan hệ được biểu diễn trên đồ thị.

By creating an adjacency matrix or defining graph edges & nodes based on relationships in dataset, can transition from simple data analysis to more sophisticated graph-based learning methods.

– Bằng cách tạo ma trận kề hoặc xác định các cạnh & nút đồ thị dựa trên các mối quan hệ trong tập dữ liệu, có thể chuyển đổi từ phân tích dữ liệu đơn giản sang các phương pháp học dựa trên đồ thị phức tạp hơn.

◦ 1.3. GNN applications: Case studies. GNNs are neural networks designed to work on relational data. They give new ways for relational data to be transformed & manipulated, by being easier to scale & more accurate than previous graph-based learning methods. In following, discuss some exciting applications of GNNs, to see, at a high level, how this class of models are solving real-world problems.

– Ứng dụng GNN: Nghiên cứu điển hình. GNN là mạng nơ-ron được thiết kế để hoạt động trên dữ liệu quan hệ. Chúng cung cấp những cách thức mới để chuyển đổi & thao tác dữ liệu quan hệ, nhờ khả năng mở rộng dễ dàng & chính xác hơn so với các phương pháp học dựa trên đồ thị trước đây. Tiếp theo, thảo luận về 1 số ứng dụng thú vị của GNN, để xem xét ở cấp độ tổng quan, cách lớp mô hình này đang giải quyết các vấn đề thực tế.

∗ 1.3.1. Recommendation engines. Enterprise graphs can exceed billions of nodes & many billions of edges. On other hand, many GNNs are benchmarked on datasets that consist of fewer than a million nodes. When applying GNNs to large graphs, adjustments of training & inference algorithms & storage techniques all have to be made. (Can learn more about specifics of scaling GNNs in Chap. 7.)

– Công cụ đề xuất. Đồ thị doanh nghiệp có thể vượt quá hàng tỷ nút & hàng tỷ cạnh. Mặt khác, nhiều GNN được đánh giá chuẩn trên các tập dữ liệu có ít hơn 1 triệu nút. Khi áp dụng GNN cho đồ thị lớn, cần phải điều chỉnh các thuật toán huấn luyện & suy luận & kỹ thuật lưu trữ. (Bạn có thể tìm hiểu thêm về các chi tiết cụ thể về việc mở rộng GNN trong Chương 7.)

1 of most well-known industry examples of GNNs is their use as recommendation engines. E.g., Pinterest is a social media platform for finding & sharing images & ideas. there are 2 major concepts to Pinterest's users: collections or categories of ideas, called *boards* (like a bulletin board); & objects a user wants to bookmark called *pins*. Pins include images, videos, & websites URLs. A user board focused on dogs might then include pins of pet photos, puppy videos, or dog-related website links. A board's pins aren't exclusive to it; a pet drawing that was pinned to Dogs board could also be pinned to a Puppies board, as shown in Fig. 1.10: A bipartite graph that is like Pinterest graph. Nodes in this case are pins & boards.

– 1 trong những ví dụ nổi tiếng nhất về GNN trong ngành là việc sử dụng chúng làm công cụ đề xuất. E.g., Pinterest là 1 nền tảng truyền thông xã hội để tìm kiếm & chia sẻ hình ảnh & ý tưởng. Có 2 khái niệm chính đối với người dùng Pinterest: bộ sưu tập hoặc danh mục ý tưởng, được gọi là *boards* (giống như bảng tin); & đối tượng mà người dùng muốn đánh dấu được gọi là *pins*. Ghim bao gồm hình ảnh, video, & URL trang web. 1 bảng người dùng tập trung vào chó có thể bao gồm các ghim ảnh thú cưng, video về chó con hoặc các liên kết trang web liên quan đến chó. Ghim của 1 bảng không chỉ giới hạn ở đó; 1 bức vẽ thú cưng được ghim vào bảng Chó cũng có thể được ghim vào bảng Chó con, như thể hiện trong Hình 1.10: Đồ thị 2 phần giống như đồ thị Pinterest. Các nút trong trường hợp này là ghim & boards.

1 way to interpret relationships between pins & boards is as a *biparite graph*. For Pinterest graph, all pins are connected to boards, but no pin is connected to another pin, & no board is connected to another board. Pins & boards are 2 classes of nodes. Members of these classes can be linked to members of other class, but not to member of same class. Pinterest graph was reported to have 3 billion nodes & 18 billion edges.

– 1 cách để diễn giải mối quan hệ giữa các chân & bảng là sử dụng đồ thị lưỡng đối. Đối với đồ thị Pinterest, tất cả các chân được kết nối với các bảng, nhưng không có chân nào được kết nối với chân khác, & không có bảng nào được kết nối với bảng khác. Chân & bảng là 2 lớp nút. Các thành viên của các lớp này có thể được liên kết với các thành viên của lớp khác, nhưng không thể liên kết với các thành viên của cùng lớp. Đồ thị Pinterest được báo cáo có 3 tỷ nút & 18 tỷ cạnh. PinSage, a graph convolutional network (GCN), was 1 of 1st documented highly scaled GNNs used in an enterprise system [9]. This was used in Pinterest's recommendation systems to overcome past challenges of applying graph-learning models to massive graphs. Compared to baseline methods, tests on this system showed it improved user engagement by 30%. Specifically, PinSage was used to predict which objects should be recommended to be included in a user's graph. However, GNNs can also be used to predict what an object is, e.g. whether it contains a dog or mountain, based on the rest of nodes in graph & how they are connected. Do a deep dive on GCNs, of which PinSage is an extension, in Chap. 3.
– PinSage, 1 mạng tích chập đồ thị (GCN), là 1 trong những GNN có quy mô lớn đầu tiên được ghi nhận sử dụng trong hệ thống doanh nghiệp [9]. Mạng này được sử dụng trong các hệ thống đề xuất của Pinterest để vượt qua những thách thức trước đây khi áp dụng các mô hình học đồ thị vào các đồ thị lớn. So với các phương pháp cơ sở, các thử nghiệm trên hệ thống này cho thấy nó đã cải thiện mức độ tương tác của người dùng lên 30%. Cụ thể, PinSage được sử dụng để dự đoán đối tượng nào nên được đề xuất đưa vào đồ thị của người dùng. Tuy nhiên, GNN cũng có thể được sử dụng để dự đoán 1 đối tượng là gì, e.g.: nó chứa 1 con chó hay 1 ngọn núi, dựa trên các nút còn lại trong đồ thị & cách chúng được kết nối. Hãy tìm hiểu sâu hơn về GCN, trong đó PinSage là 1 phần mở rộng, trong Chương 3.

∗ 1.3.2. Drug discovery & molecular science. In chemistry & molecular sciences, a prominent problem has been representing molecules in a general, application-agnostic way, & inferring possible interfaces between molecules, e.g. proteins. For molecule representation, can see: drawings of molecules that are common in high school chemistry classes bear resemblance to a graph structure, consisting of nodes (atoms) & edges (atomic bonds), as shown in Fig. 1.11: In this molecule, can see individual atoms as nodes & atomic bonds as edges.
– Khám phá thuốc & khoa học phân tử. Trong hóa học & khoa học phân tử, 1 vấn đề nổi cộm là biểu diễn phân tử theo cách tổng quát, không phụ thuộc vào ứng dụng, & suy ra các giao diện khả dĩ giữa các phân tử, e.g. protein. Về biểu diễn phân tử, có thể thấy: các hình vẽ phân tử thường thấy trong các lớp hóa học trung học phổ thông có cấu trúc đồ thị, bao gồm các nút (nguyên tử) & các cạnh (liên kết nguyên tử), như thể hiện trong Hình 1.11: Trong phân tử này, có thể thấy các nguyên tử riêng lẻ là các nút & các liên kết nguyên tử là các cạnh.
Applying GNNs to these structures can, in certain circumstances, outperform traditional "fingerprint" methods for determining properties of a molecule. These traditional methods involve creation of features by domain experts to capture a molecule's properties, e.g. interpreting presence or absence of certain molecules or atoms [10]. GNNs learn new data-driven features that can be used to group certain molecules together in new & unexpected ways or even to propose new molecules for synthesis. This is extremely important for predicting whether a chemical is toxic or safe for use or whether it has some downstream effects that can affect disease progression. Therefore, GNNs have shown themselves to be incredibly useful in field of drug discovery.
– Việc áp dụng GNN vào các cấu trúc này, trong 1 số trường hợp, có thể vượt trội hơn các phương pháp "dấu vân tay" truyền thống để xác định các đặc tính của 1 phân tử. Các phương pháp truyền thống này liên quan đến việc tạo ra các đặc điểm bởi các chuyên gia trong lĩnh vực để nắm bắt các đặc tính của 1 phân tử, e.g.: diễn giải sự hiện diện hoặc vắng mặt của 1 số phân tử hoặc nguyên tử nhất định [10]. GNN học các đặc điểm mới dựa trên dữ liệu, có thể được sử dụng để nhóm các phân tử nhất định lại với nhau theo những cách mới & bất ngờ, hoặc thậm chí đề xuất các phân tử mới để tổng hợp. Điều này cực kỳ quan trọng để dự đoán liệu 1 hóa chất có độc hại hay an toàn để sử dụng hay liệu nó có 1 số tác động hạ lưu có thể ảnh hưởng đến sự tiến triển của bệnh hay không. Do đó, GNN đã chứng tỏ mình cực kỳ hữu ích trong lĩnh vực khám phá thuốc.
Drug discovery, especially for GNNs, can be understood as a graph prediction problem. *Graph prediction* tasks are those that require learning & predicting properties about entire graph. For drug discovery, aim: predict properties e.g. toxicity or treatment effectiveness (discriminative) or to suggest entirely new graphs that should be synthesized & tested (generative). To suggest these new graphs, drug discovery methods often combine GNNs with other generative models e.g. variational graph autoencoders (VGAEs), as shown, e.g., in Fig. 1.12: A GNN system used to predict new molecules [11]. Workflow here starts on left with a representation of a molecule as a graph. In middle parts of figure, this graph representation is transformed via a GNN into a latent representation. Latent representation is then transformed back to molecule to ensure: latent space can be decoded (right). Describe VGAEs in more detail in Chap. 5 & show how we can use these to predict molecules.
– Khám phá thuốc, đặc biệt đối với GNN, có thể được hiểu là 1 bài toán dự đoán đồ thị. Nhiệm vụ *Dự đoán đồ thị* là những nhiệm vụ yêu cầu học & dự đoán các thuộc tính về toàn bộ đồ thị. Đối với khám phá thuốc, mục tiêu: dự đoán các thuộc tính, e.g. độc tính hoặc hiệu quả điều trị (phân biệt) hoặc đề xuất các đồ thị hoàn toàn mới cần được tổng hợp & thử nghiệm (sinh). Để đề xuất các đồ thị mới này, các phương pháp khám phá thuốc thường kết hợp GNN với các mô hình sinh khác, e.g. bộ mã hóa tự động đồ thị biến phân (VGAE), như được hiển thị, e.g., trong Hình 1.12: Hệ thống GNN được sử dụng để dự đoán các phân tử mới [11]. Quy trình làm việc ở đây bắt đầu ở bên trái với biểu diễn của 1 phân tử dưới dạng đồ thị. Ở phần giữa của hình, biểu diễn đồ thị này được chuyển đổi thông qua GNN thành biểu diễn tiềm ẩn. Biểu diễn tiềm ẩn sau đó được chuyển đổi trở lại thành phân tử để đảm bảo: không gian tiềm ẩn có thể được giải mã (bên phải). Mô tả VGAE chi tiết hơn trong Chương 5 & chỉ ra cách chúng ta có thể sử dụng chúng để dự đoán các phân tử.

∗ 1.3.3. Mechanical reasoning. Develop rudimentary intuition about mechanics & physics of world around us at a remarkably young age & without any formal training in subject. Do not need to write down a set of equations to know how to catch a bouncing ball. Do not even have to be in presence of a physical ball. Given a series of snapshots of a bouncing ball, can predict reasonably well where ball is going to end up.

– Lý luận cơ học. Phát triển trực giác cơ bản về cơ học & vật lý của thế giới xung quanh chúng ta ngay từ khi còn rất nhỏ & mà không cần bất kỳ sự đào tạo chính thức nào về môn học này. Không cần phải viết ra 1 loạt phương trình để biết cách bắt 1 quả bóng đang nảy. Thậm chí không cần phải ở gần 1 quả bóng thực tế. Chỉ cần 1 loạt ảnh chụp nhanh về 1 quả bóng đang nảy, bạn có thể dự đoán khá chính xác vị trí quả bóng sẽ rơi.

While these problems might seem trivial for us, they are critical for many physical industries, including manufacturing & autonomous driving. E.g., autonomous driving systems need to anticipate what will happen in a traffic scene consisting of many moving objects. Until recently, this task was typically treated as a problem of computer vision. However, more recent approaches have begun to use GNNs [12]. These GNN-based methods demonstrate: including relational information, e.g. how limbs are connected, can enable algorithms to develop physical intuition about how a person or animal moves with higher accuracy & less data.

– Mặc dù những vấn đề này có vẻ tầm thường đối với chúng ta, nhưng chúng lại rất quan trọng đối với nhiều ngành công nghiệp vật lý, bao gồm sản xuất & lái xe tự động. E.g., hệ thống lái xe tự động cần dự đoán những gì sẽ xảy ra trong 1 cảnh giao thông gồm nhiều vật thể chuyển động. Cho đến gần đây, nhiệm vụ này thường được coi là 1 vấn đề của thị giác máy tính. Tuy nhiên, các phương pháp tiếp cận gần đây hơn đã bắt đầu sử dụng GNN [12]. Các phương pháp dựa trên GNN này chứng minh: việc bao gồm thông tin quan hệ, e.g. cách các chi được kết nối, có thể cho phép các thuật toán phát triển trực giác vật lý về cách 1 người hoặc động vật di chuyển với độ chính xác cao hơn & ít dữ liệu hơn.

In Fig. 1.13: A graph representation of a mechanical body, taken from Sanchez-Gonzalez [13]. Body's segments are represented as nodes, & mechanical forces binding them are edges., give an example of how a body can be thought of as a "mechanical" graph. Input graphs for these physical reasoning systems have elements that reflect problem. E.g., when reasoning about a human or animal body, a graph could consist of nodes that represent points on body where limbs connect. For systems of free bodies, nodes of a graph could be individual objects e.g. bouncing balls. Edges of graph then represent physical relationship (e.g., gravitational forces, elastic springs, or rigid connections) between nodes. Given these inputs, GNNs learn to predict future states of a set of objects without explicitly calling on physical/mechanical laws [13]. These methods are a form of *edge prediction*, i.e., they predict how nodes connect over time. Furthermore, these models have to be dynamic to account for temporal evolution of system. Consider these problems in detail in Chap. 6.

– Trong Hình 1.13: Biểu diễn đồ thị của 1 vật thể cơ học, lấy từ Sanchez-Gonzalez [13]. Các đoạn của vật thể được biểu diễn là các nút, & lực cơ học liên kết chúng là các cạnh., đưa ra 1 ví dụ về cách 1 vật thể có thể được coi là 1 đồ thị "cơ học". Đồ thị đầu vào cho các hệ thống suy luận vật lý này có các thành phần phản ánh vấn đề. Ví dụ: khi suy luận về cơ thể người hoặc động vật, đồ thị có thể bao gồm các nút biểu diễn các điểm trên cơ thể nơi các chi kết nối. Đối với các hệ thống vật thể tự do, các nút của đồ thị có thể là các vật thể riêng lẻ, e.g.: quả bóng nảy. Các cạnh của đồ thị sau đó biểu diễn mối quan hệ vật lý (e.g.: lực hấp dẫn, lò xo đàn hồi hoặc kết nối cứng) giữa các nút. Với các đầu vào này, GNN học cách dự đoán trạng thái tương lai của 1 tập hợp các vật thể mà không cần gọi rõ ràng các định luật cơ học vật lý [13]. Các phương pháp này là 1 dạng *dự đoán cạnh*, i.e., chúng dự đoán cách các nút kết nối theo thời gian. Hơn nữa, các mô hình này phải mang tính động để tính đến sự tiến hóa theo thời gian của hệ thống. Hãy xem xét chi tiết những vấn đề này trong Chương 6.

- 1.4. When to use a GNN? Have explored real-world applications of GNNs, identify some underlying characteristics that make problems suitable for graph-based solutions. While cases of previous section clearly involved data that was naturally modeled as a graph, crucial to recognize that GNNs can also be effectively applied to problems where graph-like nature may not be immediately obvious.

  – Khi nào nên sử dụng GNN? Đã khám phá các ứng dụng thực tế của GNN, xác định 1 số đặc điểm cơ bản giúp bài toán phù hợp với các giải pháp dựa trên đồ thị. Mặc dù các trường hợp trong phần trước rõ ràng liên quan đến dữ liệu được mô hình hóa tự nhiên dưới dạng đồ thị, nhưng điều quan trọng là phải nhận ra rằng GNN cũng có thể được áp dụng hiệu quả cho các bài toán mà bản chất giống đồ thị có thể không rõ ràng ngay lập tức.

  So, instead of simply stating GNNs are useful for graph problems, this section will help you recognize patterns & relationships within your data that could benefit from graph-based modeling, even if those relationships aren't immediately apparent. Essentially, there are 3 types of criteria for identifying GNN problems: implicit relationships & interdependencies; high dimensionality & sparsity; & complex nonlocal interactions.

  – Vì vậy, thay vì chỉ đơn thuần nêu rằng GNN hữu ích cho các bài toán đồ thị, phần này sẽ giúp bạn nhận ra các mẫu & mối quan hệ trong dữ liệu của mình mà mô hình dựa trên đồ thị có thể mang lại lợi ích, ngay cả khi những mối quan hệ đó không rõ ràng ngay lập tức. Về cơ bản, có 3 loại tiêu chí để xác định các bài toán GNN: mối quan hệ ngầm & phụ thuộc lẫn nhau; tính đa chiều cao & thưa thớt; & tương tác phi cục bộ phức tạp.

  * 1.4.1. Implicit relationships & interdependencies. Graphs are versatile data structures that can model a wide range of relationships. Even when a problem doesn't initially appear to be graph-like, even if your dataset is tabular, it is beneficial to explore whether implicit relationships or interdependencies might exist that could be represented explicitly. Implicit relationships are connections that are not immediately documented or obvious within data but can still play a significant role in understanding underlying patterns & behaviors.

    – Mối quan hệ ngầm & sự phụ thuộc lẫn nhau. Đồ thị là cấu trúc dữ liệu đa năng có thể mô hình hóa 1 loạt các mối quan hệ. Ngay cả khi 1 vấn đề ban đầu có vẻ không giống đồ thị, ngay cả khi tập dữ liệu của bạn ở dạng bảng, việc khám phá xem liệu có tồn tại các mối quan hệ ngầm hoặc sự phụ thuộc lẫn nhau có thể được biểu diễn 1 cách rõ ràng hay không vẫn rất hữu ích. Mối quan hệ ngầm là những kết nối không được ghi chép ngay lập tức hoặc hiển nhiên trong dữ liệu nhưng vẫn có thể đóng 1 vai trò quan trọng trong việc hiểu các mô hình & hành vi cơ bản.

    Key indicators. To determine if your problem might benefit from modeling implicit relationships with graphs, consider whether there are hidden or indirect connections between entities in your dataset. E.g., in customer behavior analysis,

customers may appear as independent entities in a tabular dataset containing their purchases, demographics, & other details. However, they could be connected through social media influence, peer recommendations, or shared purchasing patterns, forming an underlying network of interactions.
– **Các chỉ số chính.** Để xác định xem vấn đề của bạn có thể được hưởng lợi từ việc mô hình hóa các mối quan hệ ngầm định bằng biểu đồ hay không, xem xét liệu có các kết nối ẩn hoặc gián tiếp giữa các thực thể trong tập dữ liệu của bạn hay không. Ví dụ: trong phân tích hành vi khách hàng, khách hàng có thể xuất hiện dưới dạng các thực thể độc lập trong 1 tập dữ liệu dạng bảng chứa thông tin mua hàng, nhân khẩu học & các chi tiết khác của họ. Tuy nhiên, họ có thể được kết nối thông qua ảnh hưởng trên mạng xã hội, khuyến nghị của đồng nghiệp hoặc các mô hình mua sắm chung, tạo thành 1 mạng lưới tương tác cơ bản.
Another indicator is presence of entities that share common attributes or activities without a direct or documented relationship. In case of investors, e.g., 2 or more investors may not have any formal connection but might frequently co-invest in same companies under similar conditions. Such patterns of co-investment could indicate a shared strategy or influence. In this scenario, a graph representation can be created where nodes represent individual investors, & edges are formed between nodes when 2 or more investors co-invest in the same company. Additional attributes, e.g., investment size, timing, or types of companies invested in can be added to nodes or edges, allowing GNNs to identify patterns, trends, or even potential collaboration opportunities.
– 1 chỉ báo khác là sự hiện diện của các thực thể có chung thuộc tính hoặc hoạt động mà không có mối quan hệ trực tiếp hoặc được ghi chép lại. E.g., trong trường hợp nhà đầu tư, 2 hoặc nhiều nhà đầu tư có thể không có bất kỳ mối liên hệ chính thức nào nhưng thường xuyên cùng đầu tư vào cùng 1 công ty trong các điều kiện tương tự. Các mô hình đồng đầu tư như vậy có thể chỉ ra 1 chiến lược hoặc ảnh hưởng chung. Trong trường hợp này, có thể tạo biểu đồ biểu diễn, trong đó các nút đại diện cho các nhà đầu tư cá nhân, & các cạnh được hình thành giữa các nút khi 2 hoặc nhiều nhà đầu tư cùng đầu tư vào cùng 1 công ty. Các thuộc tính bổ sung, e.g.: quy mô đầu tư, thời điểm đầu tư hoặc loại hình công ty được đầu tư, có thể được thêm vào các nút hoặc cạnh, cho phép GNN xác định các mô hình, xu hướng hoặc thậm chí các cơ hội hợp tác tiềm năng.
Additionally, consider whether data involves entities that are interconnected through shared references or co-occurrence patterns. Document & text data may not immediately suggest a graph structure, but if documents cite each other or share common topics or authors, they can be represented as nodes in a graph, with edges reflecting these relationships. Similarly, terms within documents can form co-occurrence networks, which are useful for tasks e.g. keyword extraction, document classification, or topic modeling.
– Ngoài ra, cân nhắc xem dữ liệu có bao gồm các thực thể được kết nối với nhau thông qua các tham chiếu chung hay các mẫu đồng hiện diện hay không. Dữ liệu tài liệu & văn bản có thể không gợi ý ngay lập tức 1 cấu trúc đồ thị, nhưng nếu các tài liệu trích dẫn lẫn nhau hoặc có chung chủ đề hoặc tác giả, chúng có thể được biểu diễn dưới dạng các nút trong đồ thị, với các cạnh phản ánh các mối quan hệ này. Tương tự, các thuật ngữ trong tài liệu có thể tạo thành các mạng đồng hiện diện, hữu ích cho các tác vụ như trích xuất từ khóa, phân loại tài liệu hoặc mô hình hóa chủ đề.
By identifying these key indicators in your data, you can uncover hidden or implicit relationships that can be represented explicitly through graphs. Such representations allow for more advanced analyses using GNNs, which can effectively capture & model these relationships, leading to more accurate predictions & deeper insights into data.
– Bằng cách xác định các chỉ số chính này trong dữ liệu, bạn có thể khám phá các mối quan hệ ẩn hoặc ngầm định có thể được biểu diễn rõ ràng thông qua biểu đồ. Các biểu diễn như vậy cho phép phân tích nâng cao hơn bằng cách sử dụng GNN, có thể nắm bắt hiệu quả & mô hình hóa các mối quan hệ này, dẫn đến dự đoán chính xác hơn & hiểu biết sâu sắc hơn về dữ liệu.

∗ 1.4.2. High dimensionality & sparsity. Graph-based models are particularly effective in handling high-dimensional data where many features may be sparse or missing. These models excel in situations where there are underlying structure connecting sparse entities, allowing for more meaningful analysis & improved performance.
– Các mô hình dựa trên đồ thị đặc biệt hiệu quả trong việc xử lý dữ liệu đa chiều, trong đó nhiều đặc điểm có thể thưa thớt hoặc bị thiếu. Các mô hình này đặc biệt hiệu quả trong các tình huống có cấu trúc cơ bản kết nối các thực thể thưa thớt, cho phép phân tích có ý nghĩa hơn & cải thiện hiệu suất.
Key indicators. To determine if your problem involves high-dimensional & sparse data suitable for GNNs, consider whether your dataset contains numerous entities with limited direct interactions or relationships. E.g., in recommender systems, user-item interaction data may appear tabular, but it is inherently sparse – most users only interact with a small subset of available items. By representing users & items as nodes & representing their interactions (e.g., purchases or clicks) as edges, GNNs can exploit network effects to make more accurate recommendations. These models can also address cold-start problem by uncovering both explicit & implicit relationships, leading to better performance in recommending new items to users or engaging new users with existing items.
– **Các chỉ số chính.** Để xác định xem vấn đề của bạn có liên quan đến dữ liệu đa chiều & thưa thớt phù hợp với GNN hay không, xem xét liệu tập dữ liệu của bạn có chứa nhiều thực thể với các tương tác hoặc mối quan hệ trực tiếp hạn chế hay không. Ví dụ: trong các hệ thống đề xuất, dữ liệu tương tác giữa người dùng & sản phẩm có thể xuất hiện dưới dạng bảng, nhưng bản chất của nó là thưa thớt – hầu hết người dùng chỉ tương tác với 1 tập hợp con nhỏ các sản phẩm khả dụng. Bằng cách biểu diễn người dùng & sản phẩm dưới dạng các nút & biểu diễn các tương tác của họ (e.g.: mua hàng hoặc nhấp chuột) dưới dạng các cạnh, GNN có thể khai thác hiệu ứng mạng để đưa ra các đề xuất chính xác hơn. Các mô hình này cũng có thể giải quyết vấn đề khởi động nguội bằng cách khám phá cả các mối quan hệ rõ ràng & ngầm định, dẫn đến hiệu suất tốt hơn trong việc đề xuất sản phẩm mới cho người dùng hoặc thu hút người dùng mới sử dụng các sản phẩm hiện có.

Another indicator that your problem may be suitable for graph-based models is when data represents entities that are sparsely connected but share significant characteristics. In drug discovery, e.g., molecules are represented as graphs, with atoms as nodes & chemical bonds as edges. This representation captures inherent sparsity of molecular structures, where most atoms form only a few bonds, & large portions of molecule may be distant from each other in graph. Traditional ML methods often struggle to predict properties of new molecules due to this sparsity, as they don't account for full structural context.

– 1 dấu hiệu khác cho thấy vấn đề của bạn có thể phù hợp với các mô hình dựa trên đồ thị là khi dữ liệu biểu diễn các thực thể được kết nối thưa thớt nhưng có chung các đặc điểm quan trọng. E.g., trong khám phá thuốc, các phân tử được biểu diễn dưới dạng đồ thị, với các nguyên tử là nút & các liên kết hóa học là cạnh. Cách biểu diễn này nắm bắt được tính thưa thớt vốn có của các cấu trúc phân tử, trong đó hầu hết các nguyên tử chỉ tạo thành 1 vài liên kết, & các phần lớn phân tử có thể nằm cách xa nhau trên đồ thị. Các phương pháp ML truyền thống thường gặp khó khăn trong việc dự đoán các đặc tính của các phân tử mới do tính thưa thớt này, vì chúng không tính đến toàn bộ bối cảnh cấu trúc.

Graph-based models, particularly GNNs, overcome these challenges by capturing both local atomic environments & global molecular structures. GNNs learn hierarchical features from fine-grained atomic interactions to broader molecular properties, & their ability to remain invariant to ordering of atoms ensures consistent predictions. By using graph structure of molecules, GNNs make accurate predictions from sparse, connected data, thereby accelerating drug discovery process.

– Các mô hình dựa trên đồ thị, đặc biệt là GNN, khắc phục những thách thức này bằng cách nắm bắt cả môi trường nguyên tử cục bộ & cấu trúc phân tử toàn cục. GNN học các đặc điểm phân cấp từ các tương tác nguyên tử chi tiết đến các đặc tính phân tử rộng hơn, & khả năng duy trì tính bất biến theo thứ tự nguyên tử đảm bảo các dự đoán nhất quán. Bằng cách sử dụng cấu trúc đồ thị của phân tử, GNN đưa ra các dự đoán chính xác từ dữ liệu thưa thớt & kết nối, do đó đẩy nhanh quá trình khám phá thuốc.

By recognizing these key indicators in your data, you can identify situations where graph-based models can effectively handle high-dimensional & sparse datasets. Representing such data as graphs allows GNNs to capture & use underlying structures, resulting in more accurate predictions & deeper insights across various applications.

– Bằng cách nhận diện các chỉ số chính này trong dữ liệu, bạn có thể xác định các tình huống mà mô hình dựa trên đồ thị có thể xử lý hiệu quả các tập dữ liệu đa chiều & thưa thớt. Việc biểu diễn dữ liệu dưới dạng đồ thị cho phép GNN nắm bắt & sử dụng các cấu trúc cơ bản, mang lại dự đoán chính xác hơn & hiểu biết sâu sắc hơn trên nhiều ứng dụng khác nhau.

∗ 1.4.3. Complex, nonlocal interactions. Certain problems require underlying how distant elements in a dataset influence each other. In these cases, GNNs provide a framework to capture these complex interactions, where predicted value or label of a particular data point depends not just on features of its immediate neighbors but also on those of other related data points. This capability is especially useful when relationships extend beyond direct connections to involve multiple levels or degrees of separation.

– 1 số vấn đề đòi hỏi phải hiểu rõ cách các phần tử ở xa trong 1 tập dữ liệu ảnh hưởng lẫn nhau. Trong những trường hợp này, mạng nơ-ron nhân tạo (GNN) cung cấp 1 khuôn khổ để nắm bắt những tương tác phức tạp này, trong đó giá trị dự đoán hoặc nhãn của 1 điểm dữ liệu cụ thể không chỉ phụ thuộc vào các đặc điểm của các điểm lân cận mà còn phụ thuộc vào các đặc điểm của các điểm dữ liệu liên quan khác. Khả năng này đặc biệt hữu ích khi các mối quan hệ vượt ra ngoài các kết nối trực tiếp, bao gồm nhiều cấp độ hoặc mức độ phân tách.

However, some standard GNNs, which rely primarily on local message passing, may struggle to capture long-range dependencies effectively. Advanced architectures or modifications, e.g. those incorporating global attention, nonlocal aggregation, or hierarchical message-passing, can be better address these challenges [14].

Key indicators. To determine if your problem involves complex, nonlocal interactions suitable for GNNs, consider whether outcome or behavior of 1 entity depends on attributes or actions of identities that are not directly connected to it but may be indirectly connected through other entities. E.g., in supply chain optimization, a delay in 1 supplier may not only affect its immediate downstream customers but could cascade through multiple levels of network, influencing distributors & final consumers.

– **Các chỉ số chính.** Để xác định xem vấn đề của bạn có liên quan đến các tương tác phức tạp, phi cục bộ phù hợp với GNN hay không, xem xét liệu kết quả hoặc hành vi của 1 thực thể có phụ thuộc vào các thuộc tính hoặc hành động của các danh tính không được kết nối trực tiếp với nó nhưng có thể được kết nối gián tiếp thông qua các thực thể khác hay không. E.g., trong tối ưu hóa chuỗi cung ứng, sự chậm trễ của 1 nhà cung cấp không chỉ ảnh hưởng đến các khách hàng hạ nguồn trực tiếp mà còn có thể lan truyền qua nhiều cấp độ mạng lưới, ảnh hưởng đến các nhà phân phối & người tiêu dùng cuối cùng.

Another indicator is whether problem involves scenarios where information, influence, or effects propagate through a network over time. In healthcare & epidemiology, e.g., a disease outbreak might spread from a small cluster of patients through their interactions with shared healthcare providers, common environments, or overlapping social networks. Such propagation requires an approach that captures indirect transmission pathways of information or effects.

– 1 chỉ báo khác là liệu vấn đề có liên quan đến các kịch bản mà thông tin, ảnh hưởng hoặc tác động lan truyền qua mạng lưới theo thời gian hay không. Trong chăm sóc sức khỏe & dịch tễ học, e.g., 1 đợt bùng phát dịch bệnh có thể lây lan từ 1 nhóm nhỏ bệnh nhân thông qua tương tác của họ với các nhà cung cấp dịch vụ chăm sóc sức khỏe chung, môi trường chung hoặc các mạng lưới xã hội chồng chéo. Sự lan truyền như vậy đòi hỏi 1 phương pháp tiếp cận nắm bắt các con đường truyền thông tin hoặc tác động gián tiếp.

To close this section, in determining whether your problem is a good candidate for a GNN, ask yourself these questions:

1. Are there implicit relationships or interdependencies in my data that I could model?

2. Do interactions between entities exhibit complex, nonlocal dependencies that go beyond immediate connections?
3. Is data high-dimensional & sparse, with a need to capture underlying relational structures?

If answer to any of these questions is yes, consider framing your problem as a graph & applying GNNs to unlock new insights & predictive capabilities.

– Để kết thúc phần này, khi xác định xem vấn đề của bạn có phù hợp để áp dụng mô hình mạng nơ-ron nhân tạo (GNN) hay không, tự hỏi mình những câu hỏi sau:

1. Liệu có mối quan hệ ngầm định hoặc phụ thuộc lẫn nhau nào trong dữ liệu của tôi mà tôi có thể mô hình hóa không?
2. Liệu các tương tác giữa các thực thể có biểu hiện sự phụ thuộc phức tạp, phi cục bộ, vượt ra ngoài các kết nối tức thời không?
3. Liệu dữ liệu có đa chiều & thưa thớt, cần phải nắm bắt các cấu trúc quan hệ cơ bản không?

Nếu câu trả lời cho bất kỳ câu hỏi nào trong số này là có, cân nhắc việc định hình vấn đề của bạn dưới dạng biểu đồ & áp dụng GNN để khám phá những hiểu biết mới & khả năng dự đoán.

- 1.5. Understanding how GNNs operate. In this section, explore how GNNs work, starting from initial collection of raw data to final deployment of trained models. Examine each step, highlighting processes of data handling, model building, & unique message-passing technique that sets GNNs apart from traditional DL models.
  - 1.5.1. Mental model for training a GNN. Our mental model covers data sourcing, graph representation, preprocessing, & model development workflow. Start with raw data & end up with a trained GNN model & its outputs. Fig. 1.14: Mental model of GNN project. Start with raw data, which is transformed into a graph data model that can be stored in a graph database or used in a graph processing system. From graph processing system (& some graph databases), exploratory data analysis & visualization can be done. Finally, for graph ML, data is preprocessed into a form that can be submitted for training illustrates & visualizes topics related to these stages, annotated with chaps in which these topics appear.

    – Mô hình tinh thần để huấn luyện GNN. Mô hình tinh thần của chúng tôi bao gồm việc tìm nguồn dữ liệu, biểu diễn đồ thị, tiền xử lý & quy trình phát triển mô hình. Bắt đầu với dữ liệu thô & kết thúc bằng 1 mô hình GNN đã được huấn luyện & kết quả của nó. Hình 1.14: Mô hình tinh thần của dự án GNN. Bắt đầu với dữ liệu thô, được chuyển đổi thành mô hình dữ liệu đồ thị có thể được lưu trữ trong cơ sở dữ liệu đồ thị hoặc được sử dụng trong hệ thống xử lý đồ thị. Từ hệ thống xử lý đồ thị (và 1 số cơ sở dữ liệu đồ thị), có thể thực hiện phân tích dữ liệu thăm dò & trực quan hóa. Cuối cùng, đối với học máy đồ thị, dữ liệu được tiền xử lý thành 1 biểu mẫu có thể được gửi để huấn luyện minh họa & trực quan hóa các chủ đề liên quan đến các giai đoạn này, được chú thích bằng các chương trong đó các chủ đề này xuất hiện.

    While not all workflows include every step or stage of this process, most will incorporate at least some elements. At different stages of a model development project, different parts of this process will typically be used. E.g., when *training* a model, data analysis & visualization may be needed to make design decisions, but when *deploying* a model, it may only be necessary to stream raw data & quickly preprocess it for ingestion into a model. Though this book touches on earlier stages in this mental model, bulk of book is focused on how to train different types of GNNs. When other topics are discussed, they serve to support this main focus.

    – Mặc dù không phải tất cả quy trình làm việc đều bao gồm mọi bước hoặc giai đoạn của quy trình này, nhưng hầu hết đều sẽ kết hợp ít nhất 1 số yếu tố. Ở các giai đoạn khác nhau của 1 dự án phát triển mô hình, các phần khác nhau của quy trình này thường sẽ được sử dụng. Ví dụ: khi *training* 1 mô hình, phân tích dữ liệu & trực quan hóa có thể cần thiết để đưa ra quyết định thiết kế, nhưng khi *deployment* 1 mô hình, có thể chỉ cần truyền dữ liệu thô & xử lý nhanh dữ liệu trước để đưa vào mô hình. Mặc dù cuốn sách này đề cập đến các giai đoạn trước đó của mô hình tư duy này, phần lớn nội dung sách tập trung vào cách huấn luyện các loại GNN khác nhau. Khi các chủ đề khác được thảo luận, chúng sẽ hỗ trợ cho trọng tâm chính này.

    Mental model shows core tasks of applying GNNs to ML problems, & we return to this process repeatedly through rest of book. Examine this diagram from end to end.

    – Mô hình tinh thần cho thấy các nhiệm vụ cốt lõi của việc áp dụng mạng nơ-ron nhân tạo (GNN) vào các bài toán ML, & chúng ta sẽ quay lại quá trình này nhiều lần trong suốt phần còn lại của cuốn sách. Hãy xem xét sơ đồ này từ đầu đến cuối.

    1st step in training a GNN in structuring this raw data into a graph format, if it is not already. This requires deciding which entities in data to represent as nodes & edges, as well as determining features to assign to them. Decision must also be made about data storage – whether to use a graph database, processing system, or other formats.

    – Bước đầu tiên trong quá trình huấn luyện GNN là cấu trúc dữ liệu thô này thành định dạng đồ thị, nếu dữ liệu chưa được định dạng. Điều này đòi hỏi phải quyết định những thực thể nào trong dữ liệu sẽ được biểu diễn dưới dạng nút & cạnh, cũng như xác định các đặc điểm cần gán cho chúng. Quyết định cũng cần được đưa ra về lưu trữ dữ liệu – sử dụng cơ sở dữ liệu đồ thị, hệ thống xử lý hay các định dạng khác.

    For ML, data must be preprocessed for training & inference, involving tasks e.g. sampling, batching, & splitting data into training, validation, & test sets. Throughout this book, use PyTorch Geometric (PyG), which offers specialized classes for preprocessing & data splitting while preserving graph's structure. Preprocessing is covered in most chaps, with more-in-depth explanations available in Appendix B.

    – Đối với ML, dữ liệu phải được xử lý trước để huấn luyện & suy luận, bao gồm các tác vụ như lấy mẫu, xử lý hàng loạt, & chia dữ liệu thành các tập huấn luyện, xác thực, & kiểm tra. Trong suốt cuốn sách này, sử dụng PyTorch Geometric (PyG), cung cấp các lớp chuyên biệt để xử lý trước & chia tách dữ liệu trong khi vẫn bảo toàn cấu trúc đồ thị. Phần lớn các chương đều đề cập đến tiền xử lý, với các giải thích chi tiết hơn có sẵn trong Phụ lục B.

    After processing data, can then move on to model training. In this book, cover several architectures & training types:

· Chaps. 2–3 discuss convolutional GNNs, where 1st use a GCN layer to produce graph embeddings (Chap. 2) & then train a full GCN & GraphSAGE models (Chap. 3).

· Chap. 4 explains graph attention networks (GATs), which adds attention to our GNNs.

· Chap. 5 introduces GNNs for unsupervised & generative problems, where we train & use a variational graph autoencoder (VGAE).

· Chap. 6 then explores advanced concept of spatiotemporal GNNs, based on graphs that evolve over time. Train a neural relational inference (NRI) model, which combines an autoencoder structure with a RNN.
Most of examples provided for GNNs mentioned so far are illustrated with code examples which use small-scale graphs that can fit into memory on a laptop or desktop computer.

· In Chap. 7, delve into strategies for handling data that exceeds processing capacity of a single machine.

· In Chap. 8, close with some considerations for graph & GNN projects, e.g. practical aspects of working with graph data, as well as how to convert nongraph data into a graph format.

– Sau khi xử lý dữ liệu, có thể chuyển sang huấn luyện mô hình. Trong cuốn sách này, chúng tôi sẽ đề cập đến 1 số kiến trúc & loại huấn luyện:

· Chương 2-3 thảo luận về mạng GNN tích chập, trong đó đầu tiên sử dụng 1 lớp GCN để tạo nhúng đồ thị (Chương 2) & sau đó huấn luyện 1 mô hình GCN đầy đủ & GraphSAGE (Chương 3).

· Chương 4 giải thích về mạng chú ý đồ thị (GAT), giúp tăng cường sự chú ý cho các GNN của chúng tôi.

· Chương 5 giới thiệu GNN cho các bài toán không giám sát & sinh, trong đó chúng tôi huấn luyện & sử dụng bộ mã hóa tự động đồ thị biến phân (VGAE).

· Chương 6 sau đó khám phá khái niệm nâng cao về mạng GNN không gian - thời gian, dựa trên các đồ thị phát triển theo thời gian. Huấn luyện 1 mô hình suy luận quan hệ thần kinh (NRI), kết hợp cấu trúc bộ mã hóa tự động với RNN. Hầu hết các ví dụ được cung cấp cho GNN đã đề cập cho đến nay đều được minh họa bằng các ví dụ mã sử dụng đồ thị quy mô nhỏ, có thể chứa trong bộ nhớ của máy tính xách tay hoặc máy tính để bàn.

· Trong Chương 7, đi sâu vào các chiến lược xử lý dữ liệu vượt quá khả năng xử lý của 1 máy tính.

· Trong Chương 8, kết thúc bằng 1 số cân nhắc cho các dự án đồ thị & GNN, e.g.: các khía cạnh thực tế khi làm việc với dữ liệu đồ thị, cũng như cách chuyển đổi dữ liệu không phải đồ thị sang định dạng đồ thị.

∗ 1.5.2. Unique mechanisms of a GNN model. Although there are a variety of GNN architectures at this point, they all tackle same problem of dealing with graph data in a way that is permutation invariant. They do this via encoding & exchanging information across graph structure during learning process.

– Cơ chế độc đáo của mô hình GNN. Mặc dù hiện tại có nhiều kiến trúc GNN khác nhau, tất cả đều giải quyết cùng 1 vấn đề là xử lý dữ liệu đồ thị theo cách bất biến hoán vị. Chúng thực hiện điều này thông qua việc mã hóa & trao đổi thông tin trên toàn bộ cấu trúc đồ thị trong quá trình học.

In a conventional neural network CNN, 1st need to initialize a set of parameters & functions. These include number of layers, size of layers, learning rate, loss function, batch size, & other hyperparameters. (These are all treated in detail in other books on DL, so assume familiar with those terms). Once defined these features, then train our network by iteratively updating weights of network, as shown in Fig. 1.15: Process for training a GNN, which is similar to training most other DL models.

– Trong 1 mạng nơ-ron nhân tạo thông thường (CNN), trước tiên cần khởi tạo 1 tập hợp các tham số & hàm. Chúng bao gồm số lớp, kích thước lớp, tốc độ học, hàm mất mát, kích thước lô, & các siêu tham số khác. (Tất cả những điều này đều được trình bày chi tiết trong các sách khác về DL, vì vậy coi như bạn đã quen thuộc với các thuật ngữ đó). Sau khi xác định các đặc điểm này, huấn luyện mạng bằng cách cập nhật trọng số của mạng theo từng bước, như thể hiện trong Hình 1.15: Quy trình huấn luyện GNN, tương tự như huấn luyện hầu hết các mô hình DL khác.

Explicitly, perform following steps:

1. Input our data.
2. Pass data through neural network layers that transform data according to parameters of layer & an activation rule.
3. Output a representation from final layer of network.
4. Backpropagation error, & adjust parameters accordingly.
5. Repeat these steps a fixed number of *epochs* (process by which data is passed forward & backward to train a neural network).

For tabular data, these steps are exactly as listed, as shown in Fig. 1.16: Comparison of (simple) non-GNN & GNN. GNNs have a layer that distributes data among its vertices. For graph-based or relational data, these steps are similar except that each epoch relates to 1 iteration of message passing.

– Thực hiện các bước sau 1 cách rõ ràng:

1. Nhập dữ liệu.
2. Truyền dữ liệu qua các lớp mạng nơ-ron để biến đổi dữ liệu theo các tham số của lớp & 1 quy tắc kích hoạt.
3. Xuất ra 1 biểu diễn từ lớp cuối cùng của mạng.
4. Lỗi lan truyền ngược, & điều chỉnh các tham số cho phù hợp.
5. Lặp lại các bước này với số lượng cố định *epoch* (quy trình mà dữ liệu được truyền tới & lùi để huấn luyện mạng nơ-ron).

Đối với dữ liệu dạng bảng, các bước này chính xác như được liệt kê trong Hình 1.16: So sánh (đơn giản) không phải GNN & GNN. GNN có 1 lớp phân phối dữ liệu giữa các đỉnh của nó. Đối với dữ liệu dựa trên đồ thị hoặc dữ liệu quan hệ, các bước này tương tự nhau, ngoại trừ việc mỗi epoch liên quan đến 1 lần lặp truyền thông điệp.

* 1.5.3. Message passing. *Message passing*, which is touched on throughout book, is a central mechanism in GNNs that enables nodes to communicate & share information across a graph [15]. This process allows GNNs to learn rich, informative representations of graph-structured data, which is essential for tasks e.g. node classification, link prediction, & graph-level prediction. Fig. 1.17: Elements of our message passing layer. Each message passing layer consists of an aggregation, a transformation, & an update step: 1. Input initial graph with node, edges, & features. 2. Collect all features from neighboring nodes, known as messages, for each node. 3. Aggregate messages using invariant functions e.g. sum, max, or mean. 4. Transform messages using a neural network to create new node features. 5. Update all features in graph with new node features. illustrates steps involved in typical message-passing layer.
  – Truyền tin nhắn. *Truyền tin nhắn*, được đề cập trong toàn bộ cuốn sách, là 1 cơ chế trung tâm trong GNN cho phép các nút giao tiếp & chia sẻ thông tin trên 1 đồ thị [15]. Quá trình này cho phép GNN học các biểu diễn phong phú, nhiều thông tin về dữ liệu có cấu trúc đồ thị, điều này rất cần thiết cho các tác vụ e.g. phân loại nút, dự đoán liên kết, & dự đoán cấp đồ thị. Hình 1.17: Các thành phần của lớp truyền tin nhắn của chúng tôi. Mỗi lớp truyền tin nhắn bao gồm 1 phép tổng hợp, 1 phép biến đổi, & 1 bước cập nhật: 1. Đầu vào đồ thị ban đầu với nút, cạnh, & các đặc điểm. 2. Thu thập tất cả các đặc điểm từ các nút lân cận, được gọi là các thông điệp, cho mỗi nút. 3. Tổng hợp các thông điệp bằng các hàm bất biến e.g. tổng, cực đại hoặc trung bình. 4. Biến đổi các thông điệp bằng mạng nơ-ron để tạo các đặc điểm nút mới. 5. Cập nhật tất cả các đặc điểm trong đồ thị bằng các đặc điểm nút mới. minh họa các bước liên quan đến lớp truyền tin nhắn thông thường.
  Message-passing process begins with Input (step 1) of initial graph, where every node & edge have their own features. In Collect step (step 2), each node gathers information from its immediate neighbors – these pieces of information are referred to as "messages". This step ensures that each node has access to features of its neighbors, which are crucial for understanding local graph structure. Next, in Aggregate step (step 3), collected messages from neighboring nodes are combined using an invariant function, e.g. sum, mean, or max. This aggregation consolidates information from a node's neighborhood into a single vector, capturing most relevant details about its local environment.
  – Quá trình truyền thông điệp bắt đầu với Đầu vào (bước 1) của đồ thị khởi tạo, trong đó mỗi nút & cạnh đều có các đặc trưng riêng. Trong bước Thu thập (bước 2), mỗi nút thu thập thông tin từ các nút lân cận trực tiếp của nó – những thông tin này được gọi là "thông điệp". Bước này đảm bảo rằng mỗi nút có quyền truy cập vào các đặc trưng của các nút lân cận, điều này rất quan trọng để hiểu cấu trúc đồ thị cục bộ. Tiếp theo, trong bước Tổng hợp (bước 3), các thông điệp được thu thập từ các nút lân cận được kết hợp bằng 1 hàm bất biến, e.g.: tổng, trung bình hoặc tối đa. Quá trình tổng hợp này hợp nhất thông tin từ vùng lân cận của 1 nút thành 1 vectơ duy nhất, nắm bắt các chi tiết quan trọng nhất về môi trường cục bộ của nó.
  In Transform step (step 4), aggregated messages are processed by a neural network to produce a new representation for each node. This transformation allows GNN to learn complex interactions & patterns within graph by applying nonlinear functions to aggregated information.
  – Trong bước Biến đổi (bước 4), các thông điệp tổng hợp được xử lý bởi mạng nơ-ron để tạo ra 1 biểu diễn mới cho mỗi nút. Phép biến đổi này cho phép GNN học các tương tác phức tạp & các mẫu trong đồ thị bằng cách áp dụng các hàm phi tuyến tính vào thông tin tổng hợp.
  Finally, during Update step (step 5), features of each node in graph are replaced or updated with these new representations. This completes 1 round of message passing, incorporating information from neighboring nodes to refine each node's features.
  – Cuối cùng, trong bước Cập nhật (bước 5), các đặc trưng của mỗi nút trong đồ thị được thay thế hoặc cập nhật bằng các biểu diễn mới này. Điều này hoàn tất 1 vòng truyền thông điệp, kết hợp thông tin từ các nút lân cận để tinh chỉnh các đặc trưng của từng nút.
  Each message-passing layer in a GNN allows nodes to gather information from nodes that are further away, or more "hops" away, in graph. Repeating these steps over multiple layers enables GNN to capture more complex dependencies & long-range interactions within graph.
  – Mỗi lớp truyền thông điệp trong GNN cho phép các nút thu thập thông tin từ các nút ở xa hơn, hoặc cách xa hơn "các bước nhảy", trong đồ thị. Việc lặp lại các bước này trên nhiều lớp cho phép GNN nắm bắt các mối quan hệ phụ thuộc phức tạp hơn & các tương tác tầm xa trong đồ thị.
  By using message passing, GNNs efficiently encode graph structure & data into useful representations for a variety of downstream tasks. Advanced architectures, e.g. those incorporating global attention or hierarchical message passing, further enhance model's ability to capture long-range dependencies across graph, enabling more robust performance on diverse applications.
  – Bằng cách sử dụng kỹ thuật truyền thông điệp, GNN mã hóa hiệu quả cấu trúc đồ thị & dữ liệu thành các biểu diễn hữu ích cho nhiều tác vụ hạ nguồn. Các kiến trúc tiên tiến, e.g. kiến trúc tích hợp sự chú ý toàn cục hoặc truyền thông điệp phân cấp, sẽ nâng cao hơn nữa khả năng của mô hình trong việc nắm bắt các phụ thuộc tầm xa trên toàn bộ đồ thị, cho phép hiệu suất mạnh mẽ hơn trên nhiều ứng dụng khác nhau.

◦ Summary.

* GNNs are specialized tools for handling relational, or relationship-centric, data, particularly in scenarios where traditional neural networks struggle due to complexity & diversity of graph structures.
  – GNN là công cụ chuyên dụng để xử lý dữ liệu quan hệ hoặc dữ liệu tập trung vào mối quan hệ, đặc biệt là trong các

tình huống mà mạng nơ-ron truyền thống gặp khó khăn do tính phức tạp & đa dạng của cấu trúc đồ thị.

* GNNs have found significant applications in areas e.g. recommendation engines, drug discovery, & mechanical reasoning, showcasing their versatility in handling large & complex relational data for enhanced insights & predictions.
  – GNN đã tìm thấy những ứng dụng quan trọng trong các lĩnh vực như công cụ đề xuất, khám phá thuốc, & suy luận cơ học, thể hiện tính linh hoạt của chúng trong việc xử lý dữ liệu quan hệ phức tạp & lớn để có được những hiểu biết sâu sắc & dự đoán tốt hơn.
* Specific GNN tasks include node prediction, edge prediction, graph prediction, & graph representation through embedding techniques.
  – Các nhiệm vụ cụ thể của GNN bao gồm dự đoán nút, dự đoán cạnh, dự đoán đồ thị & biểu diễn đồ thị thông qua các kỹ thuật nhúng.
* Specific GNN tasks include node prediction, edge prediction, graph prediction, & graph representation through embedding techniques.
  – Các nhiệm vụ cụ thể của GNN bao gồm dự đoán nút, dự đoán cạnh, dự đoán đồ thị & biểu diễn đồ thị thông qua các kỹ thuật nhúng.
* GNNs are best used when data is represented as a graph, indicating a strong emphasis on relationships & connections between data points. They are not ideal for individual, standalone data entries where relational information is insignificant.
  – GNN được sử dụng tốt nhất khi dữ liệu được biểu diễn dưới dạng đồ thị, thể hiện sự nhấn mạnh vào mối quan hệ & kết nối giữa các điểm dữ liệu. Chúng không lý tưởng cho các mục dữ liệu riêng lẻ, độc lập, nơi thông tin quan hệ không đáng kể.
* When deciding if a GNN solution is a good fit for your problem, consider cases that have characteristics e.g. implicit relationships, high-dimensionality, sparsity, & complex nonlocal interactions. By understanding these fundamentals, practitioners can evaluate suitability of GNNs for their specific problems, implement them effectively, & recognize their tradeoffs & limitations in real-world applications.
  – Khi quyết định xem giải pháp GNN có phù hợp với vấn đề của bạn hay không, xem xét các trường hợp có đặc điểm như mối quan hệ ngầm định, đa chiều, thưa thớt, & tương tác phi cục bộ phức tạp. Bằng cách hiểu những nguyên tắc cơ bản này, người thực hành có thể đánh giá tính phù hợp của GNN cho các vấn đề cụ thể, triển khai chúng 1 cách hiệu quả, & nhận ra những đánh đổi & hạn chế của chúng trong các ứng dụng thực tế.
* Messages passing is a core mechanism of GNN,s which enables them to encode & exchange information across a graph's structure, allowing for meaningful node, edge, & graph-level predictions. Each layer of a GNN represents 1 step of message passing, with various aggregation functions to combine messages effectively, providing insights & representations useful for ML tasks.
  – Truyền thông điệp là 1 cơ chế cốt lõi của GNN, cho phép chúng mã hóa & trao đổi thông tin trên toàn bộ cấu trúc đồ thị, cho phép đưa ra các dự đoán có ý nghĩa ở cấp độ nút, cạnh & đồ thị. Mỗi lớp của GNN đại diện cho 1 bước truyền thông điệp, với nhiều hàm tổng hợp khác nhau để kết hợp các thông điệp 1 cách hiệu quả, cung cấp thông tin chi tiết & biểu diễn hữu ích cho các tác vụ ML.

- 2. Graph embeddings. Covers:

1. Exploring graph embeddings & their importance
2. Creating node embeddings using non-GNN & GNN methods
3. Comparing node embeddings on a semi-supervised problem
4. Taking a deeper dive into embedding methods

Graph embeddings are essential tools in graph-based ML. They transform intricate structure of graphs – be it the entire graph, individual nodes, or edges – into a more manageable, lower-dimensional space. Do this to compress a complex dataset into a form that is easier to work with, without losing its inherent patterns & relationships, information to which we will apply a GNN or other ML method.

– Bao gồm:

1. Khám phá nhúng đồ thị & tầm quan trọng của chúng
2. Tạo nhúng nút bằng phương pháp không phải GNN & GNN
3. So sánh nhúng nút trong bài toán bán giám sát
4. Đi sâu hơn vào các phương pháp nhúng

Nhúng đồ thị là công cụ thiết yếu trong học máy dựa trên đồ thị. Chúng chuyển đổi cấu trúc phức tạp của đồ thị – có thể là toàn bộ đồ thị, từng nút hoặc cạnh – thành 1 không gian ít chiều hơn, dễ quản lý hơn. Thực hiện điều này để nén 1 tập dữ liệu phức tạp thành 1 dạng dễ làm việc hơn, mà không làm mất các mẫu & mối quan hệ vốn có của nó, thông tin mà chúng ta sẽ áp dụng GNN hoặc phương pháp học máy khác.

Graphs encapsulate relationships & interactions within networks, whether they are social networks, biological networks, or any system where entities are interconnected. Embeddings capture these real-life relationships in a compact from, facilitating tasks e.g. visualization, clustering, or predictive modeling.

– Đồ thị gói gọn các mối quan hệ & tương tác trong các mạng lưới, dù là mạng xã hội, mạng sinh học hay bất kỳ hệ thống nào mà các thực thể được kết nối với nhau. Nhúng nắm bắt các mối quan hệ thực tế này 1 cách cô đọng, tạo điều kiện thuận lợi cho các tác vụ như trực quan hóa, phân cụm hoặc mô hình hóa dự đoán.

There are numerous strategies to derive these embeddings, each with its unique approach & application: from classical graph algorithms that use network's topology, to linear algebra techniques that decompose matrices representing graph, & more advanced methods e.g. GNNs [1]. GNNs stand out because they can integrate embedding process directly into learning algorithm itself.

– Có nhiều chiến lược để tạo ra các nhúng này, mỗi chiến lược có cách tiếp cận & ứng dụng riêng: từ các thuật toán đồ thị cổ điển sử dụng cấu trúc mạng, đến các kỹ thuật đại số tuyến tính phân tích các ma trận biểu diễn đồ thị, & các phương pháp tiên tiến hơn, e.g. GNN [1]. GNN nổi bật vì chúng có thể tích hợp quá trình nhúng trực tiếp vào chính thuật toán học.

In traditional ML workflows, embeddings are generated as a separate step, serving as a dimensionality-reduction technique in tasks e.g. regression or classification. However, GNNs merge embedding generation with model's learning process. As network processes inputs through its layers, embeddings are refined & updated, making learning phase & embedding phase inseparable. I.e., GNNs learn most informative representative of graph data during training time.

– Trong quy trình làm việc ML truyền thống, nhúng được tạo ra như 1 bước riêng biệt, đóng vai trò là kỹ thuật giảm chiều trong các tác vụ như hồi quy hoặc phân loại. Tuy nhiên, GNN kết hợp việc tạo nhúng với quy trình học của mô hình. Khi mạng xử lý dữ liệu đầu vào qua các lớp của nó, các nhúng được tinh chỉnh & cập nhật, khiến giai đoạn học & giai đoạn nhúng trở nên không thể tách rời. Tức là, GNN học dữ liệu biểu diễn đồ thị mang tính thông tin nhất trong thời gian đào tạo.

Using graph embeddings can significantly enhance your DS & ML projects, especially when dealing with complex networked data. By capturing essence of graph in a lower-dimensional space, embeddings make it feasible to apply a variety of other ML techniques to graph data, opening up a world of possibilities for analysis & model building.

– Việc sử dụng nhúng đồ thị có thể cải thiện đáng kể các dự án DS & ML của bạn, đặc biệt là khi xử lý dữ liệu mạng phức tạp. Bằng cách nắm bắt bản chất của đồ thị trong không gian ít chiều hơn, nhúng cho phép áp dụng nhiều kỹ thuật ML khác vào dữ liệu đồ thị, mở ra 1 thế giới khả thi cho việc phân tích & xây dựng mô hình.

In this chap, begin with an introduction to graph embeddings & a case study on a graph of political book purchases. Start with Node2Vec (N2V) to establish a baseline with a non-GNN approach, guiding you through its practical application. In Sect. 2.2, shift to GNNs, offering a hands-on introduction to GNN-based embeddings, including setup, preprocessing, & visualization. Sect. 2.3 provides a comparative analysis of N2V & GNN embeddings, highlighting their applications. Chap then rounds off with a discussion of theoretical aspects of these embedding methods, with a special focus on principles behind N2V & message-passing mechanism in GNNs. Process we take in this chap is illustrated in Fig. 2.1: Summary of process & objectives in Chap. 2: 1. Preprocess Political Books dataset for embedding. 2. Use N2V & GCN to create embeddings from preprocessed data. 3. Prepare N2V embeddings & GCN embeddings for semi-supervised classification. 4. Embeddings are used as features in a random forest classifier (tabular features) &a GCN classifier (node features).

– Trong chương này, bắt đầu bằng phần giới thiệu về nhúng đồ thị & nghiên cứu điển hình về đồ thị mua sách chính trị. Bắt đầu với Node2Vec (N2V) để thiết lập đường cơ sở với phương pháp không phải GNN, hướng dẫn bạn ứng dụng thực tế. Trong Phần 2.2, chuyển sang GNN, cung cấp phần giới thiệu thực hành về nhúng dựa trên GNN, bao gồm thiết lập, tiền xử lý, & trực quan hóa. Phần 2.3 cung cấp phân tích so sánh về nhúng N2V & GNN, làm nổi bật các ứng dụng của chúng. Sau đó, chương kết thúc bằng phần thảo luận về các khía cạnh lý thuyết của các phương pháp nhúng này, đặc biệt tập trung vào các nguyên tắc đằng sau cơ chế truyền tin nhắn N2V & trong GNN. Quy trình chúng tôi thực hiện trong chương này được minh họa trong Hình 2.1: Tóm tắt các mục tiêu của quy trình & trong Chương 2: 1. Tiền xử lý tập dữ liệu Sách chính trị để nhúng. 2. Sử dụng N2V & GCN để tạo nhúng từ dữ liệu đã được xử lý trước. 3. Chuẩn bị nhúng N2V & nhúng GCN cho phân loại bán giám sát. 4. Nhúng được sử dụng làm các đặc trưng trong bộ phân loại rừng ngẫu nhiên (các đặc trưng dạng bảng) & bộ phân loại GCN (các đặc trưng dạng nút).

- 2.1. Creating embeddings with Node2Vec. Understanding relationships within a network is a core task in many fields, from social network analysis to biology & recommendation systems. In this section, explor how to create node embeddings using `Node2Vec (N2V)`, a technique inspired by `Word2Vec` from NLP [2]. N2V captures context of nodes within a graph by simulating random walks, allowing us to understand neighborhood relationships between nodes in a low-dimensional space. This approach is effective for identifying patterns, clustering similar nodes, & preparing data for ML tasks.

  – Tạo nhúng với Node2Vec. Hiểu các mối quan hệ trong mạng là 1 nhiệm vụ cốt lõi trong nhiều lĩnh vực, từ phân tích mạng xã hội đến sinh học & hệ thống khuyến nghị. Trong phần này, khám phá cách tạo nhúng nút bằng `Node2Vec (N2V)`, 1 kỹ thuật lấy cảm hứng từ `Word2Vec` trong NLP [2]. N2V nắm bắt ngữ cảnh của các nút trong đồ thị bằng cách mô phỏng các bước ngẫu nhiên, cho phép chúng ta hiểu các mối quan hệ lân cận giữa các nút trong không gian ít chiều. Phương pháp này hiệu quả trong việc xác định các mẫu, phân cụm các nút tương tự & chuẩn bị dữ liệu cho các tác vụ ML.

  To make this process accessible, use `Node2Vec` Python library, which is beginner-friendly, although it may be slower on larger graphs. N2V helps create embeddings that capture structural relationships between nodes, which we can then visualize to uncover insights about graph's structure. Our workflow involves several steps:

  1. *Load data & set N2V parameters.* Start by loading our graph data & initializing N2V with specific parameters to control random walks, e.g. walk length & number of walks per node.
  2. *Create embeddings.* N2V generates node embeddings by performing random walks on graph, effective summarizing each node's local neighborhood into a vector format.

3. *Transform embeddings.* Resulting embeddings are saved & then transformed into a format suitable for visualization.
4. *Visualize embeddings in 2D.* Use UMAP, a dimensionality reduction technique, to project these embeddings into 2D, making it easier to visualize & interpret results.

– Để quá trình này dễ hiểu hơn, sử dụng thư viện Python `Node2Vec`, thư viện này thân thiện với người mới bắt đầu, mặc dù có thể chậm hơn trên các đồ thị lớn hơn. N2V giúp tạo các nhúng nắm bắt mối quan hệ cấu trúc giữa các nút, sau đó chúng ta có thể trực quan hóa để khám phá những hiểu biết sâu sắc về cấu trúc của đồ thị. Quy trình làm việc của chúng tôi bao gồm 1 số bước:

1. *Tải dữ liệu & đặt tham số N2V.* Bắt đầu bằng cách tải dữ liệu đồ thị & khởi tạo N2V với các tham số cụ thể để kiểm soát các bước ngẫu nhiên, e.g.: độ dài bước & số lần bước trên mỗi nút.
2. *Tạo nhúng.* N2V tạo nhúng nút bằng cách thực hiện các bước ngẫu nhiên trên đồ thị, tóm tắt hiệu quả vùng lân cận cục bộ của mỗi nút thành định dạng vector.
3. *Biến đổi nhúng.* Các nhúng kết quả được lưu & sau đó được chuyển đổi thành định dạng phù hợp để trực quan hóa.
4. *Hình dung các nhúng trong 2D.* Sử dụng UMAP, 1 kỹ thuật giảm chiều, để chiếu các nhúng này vào 2D, giúp hình dung & diễn giải kết quả dễ dàng hơn.

Our data is Political Books dataset, which comprises books (nodes) connected by frequent co-purchases on Amazon.com during 2004 US election period (edges) [3]. Using this dataset provides a compelling example of how N2V can reveal underlying patterns in co-purchasing behavior, potentially reflecting broader ideological groupings among book buyers [4]. Table 2.1: Overview of Political Books dataset. provides key information about Political Books graph. The Political Books dataset contains the following:

1. Nodes: represent books about US politics sold by Amazon.com.
2. Edges: indicate frequent co-purchasing by same buyers, as suggested by Amazon's "customers who bought this book also bought these other books" feature.

In Fig. 2.2: Graph visualization of Political Books dataset. Right-leaning books (nodes) are in a lighter shade & are clustered in top half of figure, left-leaning are darker shaded circles & clustered in lower half of figure, & neural political stance are dark squares & appear in middle. When 2 nodes are connected, it indicates that they have been purchased together frequently on Amazon.com., books are shaded based on their political alignment – darker shade for liberal, lighter shade for conservative, & striped for neural. Categories were assigned by Mark Newman through a qualitative analysis of book descriptions & reviews posted on Amazon.

– Dữ liệu của chúng tôi là tập dữ liệu Sách Chính trị, bao gồm các cuốn sách (nút) được kết nối bởi các giao dịch mua chung thường xuyên trên Amazon.com trong giai đoạn bầu cử Hoa Kỳ năm 2004 (cạnh) [3]. Việc sử dụng tập dữ liệu này cung cấp 1 ví dụ thuyết phục về cách N2V có thể tiết lộ các mô hình cơ bản trong hành vi mua chung, có khả năng phản ánh các nhóm ý thức hệ rộng hơn giữa những người mua sách [4]. Bảng 2.1: Tổng quan về tập dữ liệu Sách Chính trị. cung cấp thông tin chính về biểu đồ Sách Chính trị. Tập dữ liệu Sách Chính trị chứa các thông tin sau:

1. Nút: biểu diễn các cuốn sách về chính trị Hoa Kỳ do Amazon.com bán.
2. Cạnh: biểu thị việc mua chung thường xuyên của cùng 1 người mua, như được gợi ý bởi tính năng "khách hàng đã mua cuốn sách này cũng đã mua những cuốn sách khác này" của Amazon.

Trong Hình 2.2: Biểu đồ trực quan của tập dữ liệu Sách Chính trị. Sách thiên hữu (nút) được tô sáng hơn & tập trung ở nửa trên của hình, sách thiên tả là các vòng tròn tô đậm hơn & tập trung ở nửa dưới của hình, & lập trường chính trị thần kinh là các ô vuông đậm & xuất hiện ở giữa. Khi 2 nút được kết nối, điều đó cho thấy chúng đã được mua cùng nhau thường xuyên trên Amazon.com. Sách được tô màu dựa trên khuynh hướng chính trị của chúng – tô đậm hơn cho khuynh hướng tự do, tô nhạt hơn cho khuynh hướng bảo thủ, & có sọc cho khuynh hướng thần kinh. Các danh mục được phân loại bởi Mark Newman thông qua phân tích định tính các mô tả sách & bài đánh giá được đăng trên Amazon.

This dataset, compiled by Valdis Krebs & available through GNN in Action repository or Carnegie Mellon University website, contains 105 books (nodes) & 441 edges (co-purchases). If want to learn more about background of this dataset, Krebs has written an article with this information [4].

– Bộ dữ liệu này, do Valdis Krebs biên soạn & có sẵn trên kho lưu trữ GNN in Action hoặc trang web của Đại học Carnegie Mellon, chứa 105 sách (nút) & 441 cạnh (mua chung). Nếu muốn tìm hiểu thêm về bối cảnh của bộ dữ liệu này, Krebs đã viết 1 bài báo với thông tin này [4].

Using N2V, aim to explore structure of this collection of books, uncovering insights based on political learnings & potential associations between different book categories. By visualizing embeddings created by N2V, can gain a better understanding of how books are grouped & which ones might share a common audience, providing valuable insights into consumer behavior during a politically changed period.

– Sử dụng N2V, chúng tôi hướng đến việc khám phá cấu trúc của bộ sưu tập sách này, tìm ra những hiểu biết sâu sắc dựa trên các bài học chính trị & những mối liên hệ tiềm năng giữa các thể loại sách khác nhau. Bằng cách trực quan hóa các phần nhúng do N2V tạo ra, chúng tôi có thể hiểu rõ hơn về cách sách được nhóm lại & những cuốn nào có thể có chung đối tượng độc giả, từ đó cung cấp những hiểu biết giá trị về hành vi người tiêu dùng trong giai đoạn biến động chính trị.

From visualization, data is already clustered in a logical way. This is thanks to *Kamada-Kawai algorithm* graph algorithm, which exploits topological data only without metadata & is useful for visualizing graph. This graph visualization technique positions nodes in a way that reflects their connections, aiming for an arrangement where closely connected nodes are near each other but less connected nodes are farther apart. It achieves this by treating nodes like points connected by springs,

iteratively adjusting their positions until "tension" in springs is minimized. This results in a layout that naturally reveals clusters & relationships within graph based purely on its structure.

– Từ trực quan hóa, dữ liệu đã được phân cụm theo 1 cách logic. Điều này là nhờ thuật toán đồ thị *Kamada-Kawai*, thuật toán này chỉ khai thác dữ liệu tôpô mà không cần siêu dữ liệu & rất hữu ích cho việc trực quan hóa đồ thị. Kỹ thuật trực quan hóa đồ thị này định vị các nút theo cách phản ánh kết nối của chúng, hướng đến 1 sự sắp xếp trong đó các nút có kết nối chặt chẽ sẽ ở gần nhau nhưng các nút ít kết nối hơn sẽ ở xa nhau hơn. Nó đạt được điều này bằng cách coi các nút như các điểm được kết nối bởi lò xo, điều chỉnh vị trí của chúng theo từng bước cho đến khi "lực căng" trong lò xo được giảm thiểu. Điều này dẫn đến 1 bố cục tự nhiên thể hiện các cụm & mối quan hệ trong đồ thị chỉ dựa trên cấu trúc của nó.

For Political Books dataset, Kamada-Kawai algorithm helps us visualize books (nodes) based on how often they are co-purchased on Amazon, without using any external information e.g. political alignment or book titles. This gives us an initial view of how books are grouped together by buying behavior. In next steps, use methods e.g. N2V to create embeddings that capture more detailed patterns & further distinguish different book groups.

– Đối với tập dữ liệu Sách Chính trị, thuật toán Kamada-Kawai giúp chúng tôi trực quan hóa các cuốn sách (nút) dựa trên tần suất chúng được mua chung trên Amazon, mà không cần sử dụng bất kỳ thông tin bên ngoài nào, e.g. liên kết chính trị hoặc tiêu đề sách. Điều này cung cấp cho chúng tôi cái nhìn ban đầu về cách sách được nhóm lại với nhau theo hành vi mua. Trong các bước tiếp theo, sử dụng các phương pháp, e.g. N2V, để tạo các nhúng nắm bắt các mẫu chi tiết hơn & phân biệt rõ hơn các nhóm sách khác nhau.

∗ 2.1.1. Loading data, setting parameters, & creating embeddings. Use `Node2Vec, NetworkX` libraries for our 1st hands-on encounter with graph embeddings. After installing these packages using `pip`, load our dataset's graph data, which is stored in `.gml` format (Graph Modeling Language, GML), using `NetworkX` library & generate embeddings with `Node2Vec` library.
– Đang tải dữ liệu, thiết lập tham số, & tạo nhúng. Sử dụng thư viện `Node2Vec, NetworkX` cho lần thực hành đầu tiên với nhúng đồ thị. Sau khi cài đặt các gói này bằng `pip`, tải dữ liệu đồ thị của tập dữ liệu, được lưu trữ ở định dạng `.gml` (Ngôn ngữ Mô hình Đồ thị, GML), bằng thư viện `NetworkX` & tạo nhúng bằng thư viện `Node2Vec`.
GML is a simple, human-readable plain text file format used to represent graph structures. It stores information about nodes, edges, & their attributes in a structured way, making it easy to read & write graph data. E.g., a `.gml` file might contain a list of nodes (e.g., books in our dataset) & edges (connections representing co-purchases) along with additional properties e.g. labels or weights. This format is widely used for exchanging graph data between different software & tools. By loading `.gml` file with `NetworkX`, can easily manipulate & analyze graph in Python.
– GML là 1 định dạng tệp văn bản thuần túy đơn giản, dễ đọc, được sử dụng để biểu diễn các cấu trúc đồ thị. Nó lưu trữ thông tin về các nút, cạnh & thuộc tính của chúng theo 1 cấu trúc nhất định, giúp việc đọc & ghi dữ liệu đồ thị trở nên dễ dàng. Ví dụ: tệp `.gml` có thể chứa danh sách các nút (e.g.: sách trong tập dữ liệu của chúng tôi) & các cạnh (các kết nối biểu diễn các giao dịch mua chung) cùng với các thuộc tính bổ sung, e.g.: nhãn hoặc trọng số. Định dạng này được sử dụng rộng rãi để trao đổi dữ liệu đồ thị giữa các phần mềm & công cụ khác nhau. Bằng cách tải tệp `.gml` với `NetworkX`, bạn có thể dễ dàng thao tác & phân tích đồ thị trong Python.
In `Node2Vec` library's `Node2Vec` function, can use following parameters to specify calculations done & properties of output embedding:

· *Size of embedding* (`dimensions`): Think of this as how detailed each node's profile is, as in how many different traits you are noting down. Standard detail level is 128 traits, but you can tweak this based on how complex you want each node's profile to be.

· *Length of each walk* (`Walk Length`): This is about how far each random walk through your graph goes, with 80 steps being usual journey. If want to see more of neighborhood around a node, increase this number.

· *Number of walks per node* (`Num Walks`): This tells us how many times we will take a walk starting from each node. Starting with 10 walks gives a good overview, but if you want a fuller picture of a node's surroundings, consider going on more walks.

· *Backtracking control (Return Parameter p)*: This setting helps decide if our walk should circle back to where it has been. Setting it at 1 keeps things balanced, but adjusting it can make your walks more or less exploratory.

· *Exploration Depth (In-Out Parameter q)*: This one is about choosing between taking in broader neighborhood scene (e.g.,a BFS with $q > 1$) or diving deep into specific paths (e.g., a DFS with $q < 1$), with 1 being a mix of both.

Adjust these settings based on what you are looking to understand about your nodes & their connections. Want more depth? Tweak exploration depth. Looking for broader context? Adjust walk length & number of walks. In addition, keep in mind: size of your embeddings should match level of detail you need. In general, it is a good idea to try different combinations of these parameters to see effect on embeddings.
– Trong hàm `Node2Vec` của thư viện `Node2Vec`, bạn có thể sử dụng các tham số sau để chỉ định các phép tính được thực hiện & các thuộc tính của nhúng đầu ra:

· *Kích thước nhúng* (`kích thước`): Hãy coi đây là mức độ chi tiết của hồ sơ từng nút, i.e., số lượng đặc điểm khác nhau mà bạn đang ghi lại. Mức độ chi tiết tiêu chuẩn là 128 đặc điểm, nhưng bạn có thể điều chỉnh tùy theo độ phức tạp mà bạn muốn hồ sơ của mỗi nút đạt được.

· *Độ dài mỗi lần đi* (`Độ dài lần đi`): Đây là khoảng cách mà mỗi lần đi ngẫu nhiên qua đồ thị của bạn đi được, với 80 bước là hành trình thông thường. Nếu muốn xem thêm về vùng lân cận xung quanh 1 nút, tăng con số này.

· *Số lần đi trên mỗi nút* (`Số lần đi`): Con số này cho chúng ta biết chúng ta sẽ đi bao nhiêu lần bắt đầu từ mỗi nút. Bắt đầu với 10 lần đi bộ sẽ cho 1 cái nhìn tổng quan tốt, nhưng nếu bạn muốn có cái nhìn đầy đủ hơn về môi trường xung quanh của 1 nút, cân nhắc đi bộ nhiều lần hơn.

· *Điều khiển quay lui (Tham số trả về p)*: Thiết lập này giúp quyết định xem việc đi bộ của chúng ta có nên quay lại vị trí ban đầu hay không. Đặt giá trị này ở mức 1 sẽ giữ mọi thứ cân bằng, nhưng việc điều chỉnh nó có thể khiến việc đi bộ của bạn mang tính khám phá nhiều hơn hoặc ít hơn.

· *Độ sâu khám phá (Tham số vào-ra q)*: Thiết lập này liên quan đến việc lựa chọn giữa việc xem xét toàn cảnh khu vực lân cận rộng hơn (e.g.: BFS có $q > 1$) hoặc đi sâu vào các đường dẫn cụ thể (e.g.: DFS có $q < 1$), với 1 là sự kết hợp của cả hai.

Điều chỉnh các thiết lập này dựa trên những gì bạn muốn hiểu về các nút & kết nối của chúng. Muốn có thêm chiều sâu? Hãy tinh chỉnh độ sâu khám phá. Muốn có bối cảnh rộng hơn? Hãy điều chỉnh độ dài & số lần đi bộ. Ngoài ra, nhớ rằng: kích thước nhúng phải phù hợp với mức độ chi tiết bạn cần. Nhìn chung, bạn nên thử kết hợp các thông số này theo nhiều cách khác nhau để xem hiệu quả của việc nhúng.

For this exercise, use 1st 4 parameters. Deeper details on these parameters are found in Sect. 2.4. Code in Listing 2.1: Generating N2V embeddings begins by loading graph into a variable called `book_graph`, using `read_gml` method from `NetworkX` library. Next, a N2V `model` is initialized with loaded graph. This model is set up with specific parameters: it will create 64-dimensional embeddings for each node, use walks of 30 steps along, perform 200 walks starting from each node to gather context, & run these operations in parallel across 4 workers to speed up process.

– Với bài tập này, sử dụng 4 tham số đầu tiên. Chi tiết sâu hơn về các tham số này được tìm thấy trong Mục 2.4. Mã trong Liệt kê 2.1: Tạo nhúng N2V bắt đầu bằng cách tải đồ thị vào 1 biến có tên là `book_graph`, sử dụng phương thức `read_gml` từ thư viện `NetworkX`. Tiếp theo, 1 mô hình N2V được khởi tạo với đồ thị đã tải. Mô hình này được thiết lập với các tham số cụ thể: nó sẽ tạo nhúng 64 chiều cho mỗi nút, sử dụng các bước đi 30 bước dọc theo, thực hiện 200 bước đi bắt đầu từ mỗi nút để thu thập ngữ cảnh, & chạy các thao tác này song song trên 4 worker để tăng tốc quá trình.

N2V model is then trained with additional parameters defined in `fit` method. This involves setting a context window size of 10 nodes around each target node to learn embeddings, considering all nodes at least once `min_count = 1`, & processing 4 words (nodes, in this context) each time during training.

– Mô hình N2V sau đó được huấn luyện với các tham số bổ sung được xác định trong phương thức `fit`. Phương thức này bao gồm việc thiết lập kích thước cửa sổ ngữ cảnh là 10 nút xung quanh mỗi nút mục tiêu để học các phép nhúng, xem xét tất cả các nút ít nhất 1 lần `min_count = 1`, & xử lý 4 từ (nút, trong ngữ cảnh này) mỗi lần trong quá trình huấn luyện.

Once trained, access node embeddings using `model`'s `mv` method (reflecting its NLP heritage, `wv` stands for word vectors). For our downstream tasks, we map each node to its embedding using a dictionary comprehension.

– Sau khi được đào tạo, truy cập các nhúng nút bằng phương pháp `mv` của `model` (phản ánh di sản NLP của nó, `wv` là viết tắt của các vectơ từ). Đối với các tác vụ hạ nguồn, chúng tôi ánh xạ từng nút vào nhúng của nó bằng cách sử dụng 1 thuật toán hiểu từ điển.

```
import NetworkX as nx
from Node2Vec import Node2Vec
book_graph = nx.read_gml('polbooks.gml')
node2vec = Node2Vec(book_graph, dimensions = 64, walk_length = 30, num_walks = 200, workers = 4)
model = node2vec.fit(window = 10, min_count = 1, batch_words = 4)
embeddings = {str(node) : model.wv[str(node)] for node in gml_graph.nodes()}
```

1. Loads graph data from a GML file into a NetworkX graph object
2. Initializes N2V model with specified parameters for input graph
3. Trains N2V model
4. Extracts & stores node embeddings generated by N2V model in a dictionary.

∗ 2.1.2. Demystifying embeddings. Explore what these embeddings are & why they are valuable. An *embedding* is a dense numerical vector that represents identity of a node, edge, or graph in way that captures essential information about its structure & relationships. In our context, an embedding created by N2V captures a node's position & neighborhood within graph using topological information, i.e., it summarizes how node is connected to others, effectively capturing its role & importance in network. Later, when use GNNs to create embeddings, they will also encapsulate node's features, providing an even richer representation that includes both structure & attributes. Get deeper into theoretical aspects of embedding in Sect. 2.4.

– Giải mã các phép nhúng. Khám phá các phép nhúng này là gì & tại sao chúng có giá trị. 1 phép nhúng là 1 vectơ số dày đặc biểu diễn danh tính của 1 nút, cạnh hoặc đồ thị theo cách nắm bắt thông tin thiết yếu về cấu trúc & các mối quan hệ của nó. Trong ngữ cảnh của chúng ta, 1 phép nhúng do N2V tạo ra nắm bắt vị trí & lân cận của 1 nút trong đồ thị bằng thông tin tôpô, i.e., nó tóm tắt cách nút được kết nối với các nút khác, từ đó nắm bắt hiệu quả vai trò & tầm quan trọng của nút đó trong mạng. Sau này, khi sử dụng GNN để tạo các phép nhúng, chúng cũng sẽ đóng gói các đặc trưng của nút, cung cấp 1 biểu diễn phong phú hơn, bao gồm cả cấu trúc & các thuộc tính. Tìm hiểu sâu hơn về các khía cạnh lý thuyết của phép nhúng trong Phần 2.4.

These embeddings are powerful because they transform complex, high-dimensional graph data into a fixed-size vector format that can be easily in various analyses & ML tasks. E.g., they allow us to perform exploratory data analysis by revealing patterns, clusters, & relationships within graph. Beyond this, embeddings can be directly used as features in ML models, where each dimension of vector represents a distinct feature. This is particularly useful in applications where understanding structure & connections between data points, e.g. in social networks or recommendation systems, can significantly improve model performance.

– Các phép nhúng này rất mạnh mẽ vì chúng chuyển đổi dữ liệu đồ thị phức tạp, nhiều chiều thành định dạng vector có kích thước cố định, có thể dễ dàng thực hiện trong nhiều phân tích & tác vụ ML. Ví dụ: chúng cho phép chúng ta thực hiện phân tích dữ liệu thăm dò bằng cách khám phá các mẫu, cụm, & mối quan hệ trong đồ thị. Hơn thế nữa, các phép nhúng có thể được sử dụng trực tiếp làm đặc trưng trong các mô hình ML, trong đó mỗi chiều của vector đại diện cho 1 đặc trưng riêng biệt. Điều này đặc biệt hữu ích trong các ứng dụng mà việc hiểu cấu trúc & kết nối giữa các điểm dữ liệu, e.g. trong mạng xã hội hoặc hệ thống khuyến nghị, có thể cải thiện đáng kể hiệu suất mô hình.
To illustrate, consider node representing book *Losing Bin Laden* in our Political Books dataset. Using command `model.wv['Losing Bin Laden']`, we retrieve its dense vector embedding. This vector, shown in Fig. 2.3: Extracting embedding for nodes associated with political book *Losing Bin Laden.* Output is a dense vector represented as a Python list., captures various aspects of book's role within network of co-purchased books, providing a compact, informative representation that can be used for further analysis or as input to other models.
– Để minh họa, xem xét nút biểu diễn cuốn sách `"Losing Bin Laden"` trong tập dữ liệu Sách Chính trị của chúng tôi. Sử dụng lệnh `model.wv['Losing Bin Laden']`, chúng tôi lấy được nhúng vector dày đặc của nó. Vector này, được hiển thị trong Hình 2.3: Trích xuất nhúng cho các nút liên kết với cuốn sách chính trị `"Losing Bin Laden"`. Đầu ra là 1 vector dày đặc được biểu diễn dưới dạng danh sách Python., nắm bắt các khía cạnh khác nhau về vai trò của cuốn sách trong mạng lưới các cuốn sách được mua chung, cung cấp 1 biểu diễn cô đọng, giàu thông tin, có thể được sử dụng để phân tích sâu hơn hoặc làm đầu vào cho các mô hình khác.
These embeddings can be used for exploratory data analysis to see patterns & relationships in a graph. However, their usage extends further. 1 common application is to use these vectors as features in a ML problem that uses tabular data. In that case, each element in our embedding array will become a distinct feature column in tabular data. This can add a rich representation of each node to complement other attributes in model training. In next sect, look at how to visualize these embeddings to gain deeper insights into patterns & relationships they represent.
– Các nhúng này có thể được sử dụng để phân tích dữ liệu thăm dò nhằm xem xét các mẫu & mối quan hệ trong biểu đồ. Tuy nhiên, ứng dụng của chúng còn mở rộng hơn nữa. 1 ứng dụng phổ biến là sử dụng các vectơ này làm đặc trưng trong bài toán học máy sử dụng dữ liệu bảng. Trong trường hợp đó, mỗi phần tử trong mảng nhúng của chúng ta sẽ trở thành 1 cột đặc trưng riêng biệt trong dữ liệu bảng. Điều này có thể bổ sung thêm 1 biểu diễn phong phú cho mỗi nút để bổ sung cho các thuộc tính khác trong quá trình huấn luyện mô hình. Trong phần tiếp theo, xem cách trực quan hóa các nhúng này để hiểu sâu hơn về các mẫu & mối quan hệ mà chúng biểu diễn.

* 2.1.3. Transforming & visualizing embeddings. Visualization methods e.g. Uniform Manifold Approximation & Projection (UMAP) are powerful tools for reducing high-dimensional datasets into lower-dimensional space [5]. UMAP is particularly effective for identifying inherent clusters & visualizing complex structures that are difficult to perceive in high-dimensional data. Compared to other methods, e.g. t-SNE, UMAP excels in preserving both local & global structures, making it ideal for revealing patterns & relationships across different scales in data.
– Biến đổi & trực quan hóa nhúng. Các phương pháp trực quan hóa, e.g. Xấp xỉ Đa tạp Đồng nhất & Chiếu (UMAP) là những công cụ mạnh mẽ để thu gọn các tập dữ liệu nhiều chiều thành không gian ít chiều hơn [5]. UMAP đặc biệt hiệu quả trong việc xác định các cụm cố hữu & trực quan hóa các cấu trúc phức tạp khó nhận biết trong dữ liệu nhiều chiều. So với các phương pháp khác, e.g. t-SNE, UMAP vượt trội trong việc bảo toàn cả cấu trúc cục bộ & toàn cục, khiến nó trở nên lý tưởng để khám phá các mẫu & mối quan hệ trên các quy mô khác nhau trong dữ liệu.
While N2V generates embeddings by capturing network structure of our data, UMAP takes these high-dimensional embeddings & maps them onto a lower-dimensional space (typically 2D or 3D). This mapping aims to keep similar nodes close together while also preserving broader structural relationships, providing a more comprehensive visualization of graph's topology. After obtaining our N2V embeddings & converting them into a numerical array, we initialize UMAP model with 2 components to project our data onto a 2D plane. By carefully selecting parameters e.g. number of neighbors & minimum distance. UMAP can balance between revealing fine-grained local relationships & maintaining global distances between clusters.
– Trong khi N2V tạo ra các nhúng bằng cách nắm bắt cấu trúc mạng của dữ liệu, UMAP sử dụng các nhúng đa chiều này & ánh xạ chúng vào không gian đa chiều thấp hơn (thường là 2D hoặc 3D). Việc ánh xạ này nhằm mục đích giữ các nút tương tự gần nhau đồng thời bảo toàn các mối quan hệ cấu trúc rộng hơn, cung cấp hình ảnh trực quan toàn diện hơn về cấu trúc đồ thị. Sau khi thu thập các nhúng N2V & chuyển đổi chúng thành 1 mảng số, chúng tôi khởi tạo mô hình UMAP với 2 thành phần để chiếu dữ liệu lên mặt phẳng 2D. Bằng cách lựa chọn cẩn thận các tham số, e.g.: số lượng lân cận & khoảng cách tối thiểu, UMAP có thể cân bằng giữa việc thể hiện các mối quan hệ cục bộ chi tiết & duy trì khoảng cách toàn cục giữa các cụm.
By using UMAP, gain a more accurate & interpretable visualization of our graph embeddings as shown in Listing 2.2. Visualizing embeddings using UMAP: 1. Transforms embeddings into a list of vectors for UMAP. 2. Initializes & fits UMAP. 3. Plots nodes with UMAP embeddings & color by their value. allowing us to explore & analyze patterns, clusters, & structures more effectively than with traditional methods e.g. t-SNE.
– Bằng cách sử dụng UMAP, bạn có thể có được hình ảnh trực quan chính xác hơn & dễ hiểu hơn về các nhúng đồ thị của mình như được hiển thị trong Liệt kê 2.2. Hình ảnh trực quan các nhúng bằng UMAP: 1. Chuyển đổi các nhúng thành danh sách các vectơ cho UMAP. 2. Khởi tạo & phù hợp với UMAP. 3. Vẽ các nút bằng các nhúng UMAP & tô màu theo giá trị của chúng. cho phép chúng ta khám phá & phân tích các mẫu, cụm, & cấu trúc hiệu quả hơn so với các phương pháp truyền thống, e.g.: t-SNE.
Resultant Fig. 2.4: Embeddings of Political Books dataset graph generated by N2V & visualized using UMAP. Shape & shading variations distinguish 3 political classes. encapsulates political book graph's embeddings as distilled by N2V & subsequently

visualized through UMAP. Nodes appear in different shades according to their political alignment. Visualization unfolds a discernible structure, with potential clusters that correspond to various political learnings.
– Kết quả Hình 2.4: Đồ thị nhúng của tập dữ liệu Sách Chính trị được tạo bởi N2V & trực quan hóa bằng UMAP. Hình dạng & các biến thể tô bóng phân biệt 3 lớp chính trị. đóng gói các nhúng của đồ thị sách chính trị được tinh lọc bởi N2V & sau đó được trực quan hóa bằng UMAP. Các nút xuất hiện với các sắc thái khác nhau tùy theo sự liên kết chính trị của chúng. Trực quan hóa mở ra 1 cấu trúc rõ ràng, với các cụm tiềm năng tương ứng với các bài học chính trị khác nhau.
Might wonder why don't just reduce dimensions of N2V embeddings from 64 to 2 & visualize them directly, by passing UMAP altogether? In Listing 2.3: Visualizing 2D N2V embeddings without t-SNE, show this approach, applying a 2D N2V transformation directly to our `book_graph` object.
– Có thể bạn đang thắc mắc tại sao không giảm kích thước của các nhúng N2V từ 64 xuống còn 2 & trực quan hóa chúng trực tiếp bằng cách truyền hoàn toàn UMAP? Trong Liệt kê 2.3: Trực quan hóa các nhúng N2V 2D mà không cần t-SNE, trình bày cách tiếp cận này, áp dụng phép biến đổi N2V 2D trực tiếp vào đối tượng `book_graph` của chúng ta.
`dimensions` parameter is set to 2, aiming for a direct 2D representation suitable for immediate visualization without further dimensionality reduction. Other parameters are kept the same.
– Tham số `dimensions` được đặt thành 2, hướng đến biểu diễn 2D trực tiếp, phù hợp để trực quan hóa ngay lập tức mà không cần giảm kích thước thêm. Các tham số khác được giữ nguyên.
Once model is fitted with specified window & word batch settings, extract 2D embeddings & store them in a dictionary keyed by string representation of each node. This enables a direct mapping from node to its embedding vector.
– Sau khi mô hình được điều chỉnh với các thiết lập hàng loạt cửa sổ & từ được chỉ định, trích xuất các nhúng 2D & lưu trữ chúng trong 1 từ điển được khóa bằng chuỗi biểu diễn của mỗi nút. Điều này cho phép ánh xạ trực tiếp từ nút đến vectơ nhúng của nó.
Extracted 2D points are compiled into a NumPy array & plotted. Use standard Matplotlib library to create a scatterplot of these points using prepared color scheme to represent political leaning of each node visually.
– Các điểm 2D được trích xuất sẽ được biên dịch thành 1 mảng NumPy & vẽ đồ thị. Sử dụng thư viện Matplotlib chuẩn để tạo biểu đồ phân tán các điểm này bằng cách sử dụng bảng màu đã chuẩn bị sẵn để thể hiện trực quan khuynh hướng chính trị của từng nút.
Outcome shows how books are separated by political leanings, similar to UMAP result, but where books are more bunched together Fig. 2.5: Embeddings of Political Books dataset graph generated & visualized by N2V for 2D. Shape & shading variations distinguish 3 political classes. Here, see a similar clustering by political leaning as earlier in Fig. 2.5 but more bunched together. 2 embeddings are then shown in Fig. 2.6: Comparison of embeddings generated by N2V & t-SNE & a direct visualization of 2D Node2Vec.
– Kết quả cho thấy sách được phân loại theo khuynh hướng chính trị, tương tự như kết quả UMAP, nhưng sách được gom lại với nhau nhiều hơn Hình 2.5: Đồ thị nhúng của tập dữ liệu Sách Chính trị được tạo & trực quan hóa bằng N2V cho 2D. Các biến thể hình & tô bóng phân biệt 3 lớp chính trị. Ở đây, xem 1 phân cụm tương tự theo khuynh hướng chính trị như trước đó trong Hình 2.5 nhưng được gom lại với nhau nhiều hơn. 2 nhúng sau đó được hiển thị trong Hình 2.6: So sánh các nhúng được tạo bởi N2V & t-SNE & trực quan hóa trực tiếp của Node2Vec 2D.
Clear: both methods know to separate books into groups based on political leanings. N2V is less expressive in how it separates books, bunching them together across 2D. Meanwhile, UMAP is better for spreading out books in 2D. Relevant benefit or information contained within these dimensions depends on task at hand.
– Rõ ràng: cả 2 phương pháp đều biết cách phân loại sách thành các nhóm dựa trên khuynh hướng chính trị. N2V kém biểu cảm hơn trong cách phân loại sách, gom chúng lại với nhau trên không gian 2 chiều. Trong khi đó, UMAP tốt hơn trong việc phân bổ sách trên không gian 2 chiều. Lợi ích hoặc thông tin liên quan chứa trong các chiều này phụ thuộc vào nhiệm vụ được giao.

* 2.1.4. Beyond visualization: Applications & considerations of N2V embeddings. While visualizing N2V embeddings offers intuitive insights into dataset's structure, their usage extends far beyond graphical representation. N2V is an embedding method designed specifically for graphs; it captures both local & global structural properties of nodes by simulating random walks through graph. This process allows N2V to create dense, numerical vectors that summarize position & context of each node within overall network.
– Vượt ra ngoài trực quan hóa: Ứng dụng & cân nhắc về nhúng N2V. Mặc dù trực quan hóa nhúng N2V mang lại cái nhìn sâu sắc trực quan về cấu trúc của tập dữ liệu, nhưng ứng dụng của chúng còn vượt xa việc biểu diễn đồ họa. N2V là 1 phương pháp nhúng được thiết kế riêng cho đồ thị; nó nắm bắt cả các thuộc tính cấu trúc cục bộ & toàn cục của các nút bằng cách mô phỏng các bước ngẫu nhiên qua đồ thị. Quá trình này cho phép N2V tạo ra các vectơ số dày đặc, tóm tắt vị trí & ngữ cảnh của mỗi nút trong toàn bộ mạng.
These embeddings can then serve as feature-rich inputs for a variety of ML tasks, e.g. classification, recommendation, or clustering. E.g. in our Political Books dataset, embeddings could help predict a book's political learning based on its co-purchase patterns or could recommend books to users with similar political interests. They might even be used to forecast future sales based on content of a book.
– Những nhúng này sau đó có thể đóng vai trò là dữ liệu đầu vào giàu tính năng cho nhiều tác vụ ML, e.g. phân loại, đề xuất hoặc phân cụm. Ví dụ: trong tập dữ liệu Sách Chính trị của chúng tôi, các nhúng có thể giúp dự đoán khả năng học hỏi chính trị của 1 cuốn sách dựa trên các mô hình mua chung hoặc có thể đề xuất sách cho người dùng có cùng sở thích chính trị. Chúng thậm chí có thể được sử dụng để dự báo doanh số bán hàng trong tương lai dựa trên nội dung của 1 cuốn sách.
However, important to understand nature of N2V's learning approach, which is *transductive*. Transductive learning is

designed to work only with specific dataset it was trained on & cannot generalize to new, unseen nodes without retraining model. This characteristic makes N2V highly effective for static datasets where all nodes & edges are known up front but less suitable for dynamic settings where new data points or connections frequently appear. Essentially, N2V focuses on extracting detailed patterns & relationships from existing graph rather than developing a model that can easily adapt to new data.

– Tuy nhiên, điều quan trọng là phải hiểu bản chất của phương pháp học tập của N2V, đó là *transductive*. Học tập transductive được thiết kế để chỉ hoạt động với tập dữ liệu cụ thể mà nó được huấn luyện & không thể khái quát hóa sang các nút mới, chưa được biết đến mà không cần huấn luyện lại mô hình. Đặc điểm này khiến N2V rất hiệu quả đối với các tập dữ liệu tĩnh trong đó tất cả các nút & cạnh đều được biết trước nhưng ít phù hợp hơn với các thiết lập động, nơi các điểm dữ liệu hoặc kết nối mới thường xuyên xuất hiện. Về cơ bản, N2V tập trung vào việc trích xuất các mẫu & mối quan hệ chi tiết từ đồ thị hiện có thay vì phát triển 1 mô hình có thể dễ dàng thích ứng với dữ liệu mới.

While this transductive nature has its limitations, it also offer significant advantages. Because n2V uses full structure of graph during training, it can capture intricate relationships & dependencies that might be missed by more generalized methods. This makes N2V particularly powerful for tasks where complete, fixed structure of data is known & stable. However, to apply N2V effectively, it is crucial to ensure that graph data is represented in a way that captures all relevant features. In some cases, additional edges or nodes may need to be added to graph to fully represent underlying relationships.

– Mặc dù bản chất chuyển đổi này có những hạn chế, nhưng nó cũng mang lại những lợi thế đáng kể. Vì n2V sử dụng toàn bộ cấu trúc đồ thị trong quá trình huấn luyện, nó có thể nắm bắt được các mối quan hệ phức tạp & các mối phụ thuộc mà các phương pháp tổng quát hơn có thể bỏ sót. Điều này làm cho N2V đặc biệt mạnh mẽ đối với các tác vụ mà cấu trúc dữ liệu hoàn chỉnh, cố định đã biết & ổn định. Tuy nhiên, để áp dụng N2V hiệu quả, điều quan trọng là phải đảm bảo dữ liệu đồ thị được biểu diễn theo cách nắm bắt được tất cả các đặc điểm liên quan. Trong 1 số trường hợp, có thể cần thêm các cạnh hoặc nút bổ sung vào đồ thị để biểu diễn đầy đủ các mối quan hệ cơ bản.

For those interested in a deeper understanding of transductive models & how N2V's approach compares to other methods, further details are provided in Sect. 2.4.2, which explore tradeoffs between transductive & inductive learning in greater depth [6, 7], helping you understand when each approach is most appropriate.

– Đối với những người quan tâm đến việc hiểu sâu hơn về các mô hình chuyển đổi & cách tiếp cận của N2V so sánh với các phương pháp khác, các chi tiết bổ sung được cung cấp trong Phần 2.4.2, khám phá sự đánh đổi giữa học chuyển đổi & quy nạp sâu hơn [6, 7], giúp bạn hiểu khi nào thì mỗi cách tiếp cận là phù hợp nhất.

While N2V is effective for generating embeddings that capture structure of a fixed graph, real-world data often demands a more flexible & generalizable approach. This need brings us to our 1st GNN architecture for creating node embeddings. Unlike N2V, which is a transductive method limited to specific nodes & edges in training data, GNNs can learn in an *inductive* manner, i.e., GNNs are capable of generalizing to new, unseen nodes or edges without requiring retraining on entire graph.

– Mặc dù N2V hiệu quả trong việc tạo ra các phép nhúng nắm bắt cấu trúc của 1 đồ thị cố định, dữ liệu thực tế thường đòi hỏi 1 phương pháp linh hoạt hơn & có thể khái quát hóa. Nhu cầu này đưa chúng ta đến kiến trúc GNN đầu tiên để tạo ra các phép nhúng nút. Không giống như N2V, vốn là phương pháp chuyển đổi giới hạn ở các nút & cạnh cụ thể trong dữ liệu huấn luyện, GNN có thể học theo cách *quy nạp*, i.e., GNN có khả năng khái quát hóa sang các nút hoặc cạnh mới, chưa được biết đến mà không cần phải huấn luyện lại toàn bộ đồ thị.

GNNs achieve this by not only understanding network's complex structure but also by incorporating node features & relationships into learning process. This approach allows GNNs to adapt dynamically to changes in graph, making them well-suited for applications where data is continually evolving. Shift from N2V to GNNs represents a key transition from focusing on deep analysis within a static dataset to a broader applicability across diverse, evolving networks. This adaptability sets stage for a wider range of graph-based ML applications that require flexibility & scalability. In next sect, explore how GNNs go beyond capabilities of N2V & other transductive methods, allowing for ore versatile & powerful models that can handle dynamic nature of real-world data.

– GNN đạt được điều này không chỉ bằng cách hiểu cấu trúc phức tạp của mạng mà còn bằng cách kết hợp các đặc điểm nút & mối quan hệ vào quá trình học. Cách tiếp cận này cho phép GNN thích ứng linh hoạt với những thay đổi trong đồ thị, khiến chúng phù hợp với các ứng dụng mà dữ liệu liên tục phát triển. Sự chuyển đổi từ N2V sang GNN thể hiện 1 bước chuyển đổi quan trọng từ việc tập trung vào phân tích sâu trong 1 tập dữ liệu tĩnh sang khả năng ứng dụng rộng rãi hơn trên các mạng lưới đa dạng & đang phát triển. Khả năng thích ứng này đặt nền tảng cho 1 loạt các ứng dụng học máy dựa trên đồ thị đòi hỏi tính linh hoạt & khả năng mở rộng. Trong phần tiếp theo, khám phá cách GNN vượt ra ngoài khả năng của N2V & các phương pháp chuyển đổi khác, cho phép tạo ra các mô hình linh hoạt & mạnh mẽ hơn có thể xử lý bản chất động của dữ liệu thực tế.

- 2.2. Creating embeddings with a GNN. While N2V provides a powerful method for generating embeddings by capturing local & global structure of a graph, it is fundamentally a transductive approach, i.e., it cannot easily generalize to unseen nodes or edges without retraining. Although there have been extensions to N2V that enable it to work in inductive settings, GNNs are inherently designed for inductive learning. I.e., they can learn general patterns from graph data that allow them to make predictions or to generate embeddings for new nodes or edges without need to retrain entire model. This gives GNNs a significant edge in scenarios where flexibility & adaptability are crucial.

  – Tạo nhúng với GNN. Mặc dù N2V cung cấp 1 phương pháp mạnh mẽ để tạo nhúng bằng cách nắm bắt cấu trúc cục bộ & toàn cục của đồ thị, nhưng về cơ bản, nó là 1 phương pháp chuyển đổi, i.e., nó không thể dễ dàng khái quát hóa sang các

nút hoặc cạnh chưa biết mà không cần đào tạo lại. Mặc dù đã có các phần mở rộng cho N2V cho phép nó hoạt động trong các thiết lập quy nạp, GNN vốn được thiết kế cho việc học quy nạp. Tức là, chúng có thể học các mẫu chung từ dữ liệu đồ thị, cho phép chúng đưa ra dự đoán hoặc tạo nhúng cho các nút hoặc cạnh mới mà không cần đào tạo lại toàn bộ mô hình. Điều này mang lại cho GNN 1 lợi thế đáng kể trong các tình huống mà tính linh hoạt & khả năng thích ứng là rất quan trọng.

GNNs not only incorporate structural information of graph, like N2V, but they also use node features to create richer representations. This dual capacity allows GNNs to learn both complex relationships within graph & specific characteristics of individual nodes, enabling them to excel in tasks where both types of information are important.

– GNN không chỉ kết hợp thông tin cấu trúc của đồ thị, như N2V, mà còn sử dụng các đặc điểm của nút để tạo ra các biểu diễn phong phú hơn. Khả năng kép này cho phép GNN học cả các mối quan hệ phức tạp trong đồ thị & các đặc điểm cụ thể của từng nút, cho phép chúng vượt trội trong các tác vụ mà cả 2 loại thông tin đều quan trọng.

That said, while GNNs have demonstrated impressive performance across many applications, they do not universally outperform methods e.g. N2V in all cases. E.g., N2V & other random walk-based methods can sometimes perform better in scenarios where labeled data is scarce or noisy, thanks to their ability to work with just graph structure without needing additional node features.

– Tuy nhiên, mặc dù GNN đã chứng minh hiệu suất ấn tượng trong nhiều ứng dụng, chúng không phải lúc nào cũng vượt trội hơn các phương pháp như N2V trong mọi trường hợp. E.g., N2V & các phương pháp dựa trên bước đi ngẫu nhiên khác đôi khi có thể hoạt động tốt hơn trong các tình huống dữ liệu được gắn nhãn khan hiếm hoặc nhiễu, nhờ khả năng hoạt động chỉ với cấu trúc đồ thị mà không cần thêm các đặc trưng nút.

∗ 2.2.1. Constructing embeddings. Unlike N2V, GNNs learn graph representations & perform tasks e.g. node classification or link prediction simultaneously during training. Information from entire is processed through successive GNN layers, each refining node embeddings without requiring a separate step for their creation.

– Xây dựng nhúng. Không giống như N2V, GNN học các biểu diễn đồ thị & thực hiện đồng thời các tác vụ như phân loại nút hoặc dự đoán liên kết trong quá trình huấn luyện. Thông tin từ toàn bộ được xử lý qua các lớp GNN kế tiếp, mỗi lớp tinh chỉnh các nhúng nút mà không cần 1 bước riêng biệt để tạo chúng.

To demonstrate how a GNN extracts features from graph data, perform a straightforward pass-through using an untrained model to generate preliminary embeddings. Even without optimization typically involved in training, this approach will show how GNNs use message passing to update embeddings, capturing both graph's structure & its node features. When optimization is added, these embeddings become tailored to specific tasks e.g. node classification or link prediction.

– Để minh họa cách GNN trích xuất các đặc điểm từ dữ liệu đồ thị, thực hiện 1 phép truyền trực tiếp đơn giản bằng 1 mô hình chưa được huấn luyện để tạo ra các nhúng sơ bộ. Ngay cả khi không có quá trình tối ưu hóa thường được sử dụng trong huấn luyện, phương pháp này vẫn sẽ cho thấy cách GNN sử dụng việc truyền thông điệp để cập nhật các nhúng, nắm bắt cả cấu trúc của đồ thị & các đặc điểm nút của nó. Khi được tối ưu hóa, các nhúng này sẽ được điều chỉnh cho phù hợp với các tác vụ cụ thể, e.g. phân loại nút hoặc dự đoán liên kết.

· Defining our GNN architecture. Initiate our process by defining a siple GCN architecture, as shown in Listing 2.4: `SimpleGNN` class. Our `SimpleGNN` class inherits from `torch.nn.Module` & is composed of 2 `GCN-Conv` layers, which are building blocks of our GNN. This architecture is shown in Fig. 2.7: Architecture diagram of `SimpleGNN` model, consisting of 1st layer, a message passing layer `self.conv1`, an activation `torch.relu`, a dropout layer `torch.dropout`, & a 2nd message passing layer.

– Định nghĩa kiến trúc GNN của chúng ta. Bắt đầu quy trình bằng cách định nghĩa 1 kiến trúc GCN đơn, như được hiển thị trong Liệt kê 2.4: `SimpleGNN` class. Lớp `SimpleGNN` của chúng ta kế thừa từ `torch.nn.Module` & bao gồm 2 lớp `GCN-Conv`, là các khối xây dựng nên GNN của chúng ta. Kiến trúc này được hiển thị trong Hình 2.7: Sơ đồ kiến trúc của mô hình `SimpleGNN`, bao gồm lớp thứ nhất, lớp truyền tin nhắn `self.conv1`, lớp kích hoạt `torch.relu`, lớp dropout `torch.dropout`, & lớp truyền tin nhắn thứ hai.

Talk about architecture aspects specific to GNNs. Activation & dropout are common in many DL scenarios. GNN layers, however, are different from conventional DL layers in a fundamental way. Core principle that allows GNNs to learn from graph data is message passing. For each GNN layer, in addition to updating layer's weights, a "message" is gathered from every node or edge neighborhood & used to update an embedding. Essentially, each node sends messages to its neighbors & simultaneously receives messages from them. For every node, its new embedding is computed by combining its own features with aggregated messages from its neighbors, through a combination of nonlinear transformations.

– Nói về các khía cạnh kiến trúc đặc thù của GNN. Activation & dropout là phổ biến trong nhiều kịch bản DL. Tuy nhiên, các lớp GNN khác biệt cơ bản so với các lớp DL thông thường. Nguyên lý cốt lõi cho phép GNN học từ dữ liệu đồ thị là truyền thông điệp. Đối với mỗi lớp GNN, ngoài việc cập nhật trọng số của lớp, 1 "thông điệp" được thu thập từ mọi nút hoặc lân cận cạnh & được sử dụng để cập nhật nhúng. Về cơ bản, mỗi nút gửi thông điệp đến các nút lân cận & đồng thời nhận thông điệp từ chúng. Đối với mỗi nút, nhúng mới của nó được tính toán bằng cách kết hợp các đặc trưng của chính nó với các thông điệp tổng hợp từ các nút lân cận, thông qua sự kết hợp của các phép biến đổi phi tuyến tính.

In this example, we are going to be using a graph convolutional network (GCN) to act as our message-passing GNN layers. Describe GCNs in much more detail i Chap. 3. For now, just need to know that GCNs act as message-passing layers that are critical in constructing embeddings.

– Trong ví dụ này, chúng ta sẽ sử dụng mạng tích chập đồ thị (GCN) để hoạt động như các lớp GNN truyền thông điệp. Mô tả chi tiết hơn về GCN trong Chương 3. Hiện tại, chúng ta chỉ cần biết rằng GCN hoạt động như các lớp truyền thông điệp, đóng vai trò quan trọng trong việc xây dựng các hàm nhúng.

· Data preparation. Next, prepare our data. Start with same graph from previous section, `books_gml`, in its `NetworkX` form. Have to convert this `NetworkX` object into a tensor form that is suitable to use with PyTorch operations. Because PyTorch Geometric (PyG) has many functions that convert graph objects, we can do this quite simply with `data = from_NetworkX(gml_graph)`. Method `from_NetworkX` specifically translates edge lists & node.edge attributes into PyTorch tensors.
– Chuẩn bị dữ liệu. Tiếp theo, chuẩn bị dữ liệu. Bắt đầu với cùng 1 đồ thị từ phần trước, `books_gml`, ở dạng `NetworkX`. Phải chuyển đổi đối tượng `NetworkX` này sang dạng tensor phù hợp để sử dụng với các phép toán PyTorch. Vì PyTorch Geometric (PyG) có nhiều hàm chuyển đổi đối tượng đồ thị, chúng ta có thể thực hiện việc này khá đơn giản với `data = from_NetworkX(gml_graph)`. Phương thức `from_NetworkX` đặc biệt chuyển đổi các thuộc tính danh sách cạnh & node.edge thành tensor PyTorch.
For GNNs, generating node embeddings requires initializing node features. In our case, we do not have any predefined node features. When no node features are available or they are not informative, it is common practice to initialize node features randomly. A more effective approach: use *Xavier initialization*, which sets initial node features with values drawn from a distribution that keeps variety of activations consistent across layers. This technique ensures: model starts with a balanced representation, preventing problems e.g. vanishing or exploding gradients.
– Đối với GNN, việc tạo nhúng nút đòi hỏi phải khởi tạo các đặc trưng nút. Trong trường hợp của chúng tôi, chúng tôi không có bất kỳ đặc trưng nút nào được xác định trước. Khi không có đặc trưng nút nào khả dụng hoặc chúng không cung cấp thông tin, việc khởi tạo các đặc trưng nút 1 cách ngẫu nhiên là 1 phương pháp phổ biến. 1 cách tiếp cận hiệu quả hơn: sử dụng *Xavier initialization*, phương pháp này đặt các đặc trưng nút ban đầu với các giá trị được lấy từ 1 phân phối duy trì tính nhất quán của các phép kích hoạt trên các lớp. Kỹ thuật này đảm bảo: mô hình bắt đầu với 1 biểu diễn cân bằng, ngăn ngừa các vấn đề như gradient biến mất hoặc bùng nổ.
By initializing `data.x` with Xavier initialization, provide GNN with a starting point that allows it to learn meaningful node embeddings from noninformative features. During training, network adjusts these initial values to minimize loss function. When loss function is aligned with a specific target, e.g. node prediction, embeddings learned from initial random features will become tailored to task at hand, resulting in more effective representations. Randomize node features using following:
– Khởi tạo, cung cấp cho GNN 1 điểm khởi đầu cho phép nó học các phép nhúng nút có ý nghĩa từ các đặc trưng không mang tính thông tin. Trong quá trình huấn luyện, mạng sẽ điều chỉnh các giá trị ban đầu này để giảm thiểu hàm mất mát. Khi hàm mất mát được căn chỉnh với 1 mục tiêu cụ thể, e.g.: dự đoán nút, các phép nhúng học được từ các đặc trưng ngẫu nhiên ban đầu sẽ được điều chỉnh cho phù hợp với tác vụ hiện tại, mang lại hiệu quả biểu diễn cao hơn. Ngẫu nhiên hóa các đặc trưng nút bằng cách sử dụng các bước sau:

```
1  data.x = torch.randn((data.num_nodes, 64), dtype = torch.float)
2  'nn.init.xavier_uniform_(data.x) '
```

We could have also used embeddings from N2V exercise to use as node features. Recall `node_embeddings` object from Sect. 2.1.3:
– Chúng ta cũng có thể sử dụng các nhúng từ bài tập N2V để làm đặc trưng nút. Hãy nhớ lại đối tượng `node_embeddings` từ Mục 2.1.3:

```
node_embeddings = [embeddings[str(node)] for node in gml_graph.nodes()]
```

From this, can convert node embedding to a PyTorch tensor object & assign it to node feature object, `data.x`:
– Từ đó, có thể chuyển đổi nhúng nút thành đối tượng tenxơ PyTorch & gán nó cho đối tượng tính năng nút, `data.x`:

```
node_features = torch.tensor(node_embedding, dtype = torch.float)
data.x = node_features
```

· Passing graph through GNN. With structure of our GNN model defined & our graph data formatted for PyG, proceed to embedding generation step. Initialize our model, `SimpleGNN`, specifying number of features for each node & size of hidden channels within network.
– Truyền đồ thị qua GNN. Với cấu trúc mô hình GNN đã được xác định & dữ liệu đồ thị được định dạng cho PyG, tiến hành bước tạo nhúng. Khởi tạo mô hình `SimpleGNN`, chỉ định số lượng đặc trưng cho mỗi nút & kích thước của các kênh ẩn trong mạng.

```
model = SimpleGNN(num_features = data.x.shape[1], hidden_channels = 64)
```

Here, specify 64 hidden channels because we want to compare resulting embeddings to the ones we produced using `node2vec` method, which had 64 dimensions. Because 2nd GNN layer is last layer, output will be a 64-element vector.
– Ở đây, ta chỉ định 64 kênh ẩn vì chúng ta muốn so sánh các nhúng kết quả với các nhúng chúng ta tạo ra bằng phương pháp `node2vec`, có 64 chiều. Vì lớp GNN thứ 2 là lớp cuối cùng, đầu ra sẽ là 1 vectơ 64 phần tử.
Once initialized, we switch model to evaluation mode using `model.eval()`. This mode is used during inference or validation phases when we want to make predictions or assess model performance without modifying model's parameters. Specifically, `model.eval()` turns off certain behaviors specific to training, e.g. *dropout*, which randomly deactivates some neurons to prevent overfitting, & *batch normalization*, which normalizes inputs across a mini-batch. By disabling these features, model provides consistent & deterministic outputs, ensuring: evaluation accurately reflects its true performance on unseen data.

– Sau khi khởi tạo, chúng tôi chuyển mô hình sang chế độ đánh giá bằng `model.eval()`. Chế độ này được sử dụng trong các giai đoạn suy luận hoặc xác thực khi chúng tôi muốn đưa ra dự đoán hoặc đánh giá hiệu suất mô hình mà không cần sửa đổi các tham số của mô hình. Cụ thể, `model.eval()` sẽ tắt 1 số hành vi cụ thể trong quá trình huấn luyện, e.g.: `dropout`, vô hiệu hóa ngẫu nhiên 1 số nơ-ron để ngăn ngừa quá khớp, & `batch normalization`, chuẩn hóa đầu vào trên 1 mini-batch. Bằng cách vô hiệu hóa các tính năng này, mô hình cung cấp đầu ra nhất quán & xác định, đảm bảo: đánh giá phản ánh chính xác hiệu suất thực tế của nó trên dữ liệu chưa được biết đến.
Important to disable gradient computations because they are not necessary for forward pass & embedding generation. So, we employ `torch.no_grad()`, which ensures: computational graph that records operations for backpropagation is not constructed, preventing us from accidentally changing performance.
– Điều quan trọng là phải tắt tính toán gradient vì chúng không cần thiết cho việc tạo nhúng & truyền tiếp. Vì vậy, chúng tôi sử dụng `torch.no_grad()`, đảm bảo: đồ thị tính toán ghi lại các thao tác cho lan truyền ngược không được xây dựng, ngăn chúng tôi vô tình thay đổi hiệu suất.
Next, pass our node-feature matrix `data.x` & edge index `data.edge_index` through model. Result is `gnn_embeddings`, a tensor where each row corresponds to embedding of a node in our graph – a numerical representation learned by our GNN, ready for downstream tasks e.g. visualization or classification:
– Tiếp theo, truyền ma trận đặc trưng nút `data.x` & chỉ số cạnh `data.edge_index` qua mô hình. Kết quả là `gnn_embeddings`, 1 tenxơ trong đó mỗi hàng tương ứng với việc nhúng 1 nút vào đồ thị của chúng ta – 1 biểu diễn số được học bởi GNN, sẵn sàng cho các tác vụ tiếp theo, e.g. trực quan hóa hoặc phân loại:

```
model.eval()
with torch.no_grad():
    gnn_embeddings = model(data.x, data.edge_index)
```

After producing these embeddings, use UMAP to visualize them. Since we have been working with PyTorch tensor data types running on a GPU, need to convert our embeddings to a NumPy array data type to use analysis methods outside of PyTorch, which are done on a CPU:
– Sau khi tạo các nhúng này, sử dụng UMAP để trực quan hóa chúng. Vì chúng ta đang làm việc với các kiểu dữ liệu tensor PyTorch chạy trên GPU, cần chuyển đổi các nhúng sang kiểu dữ liệu mảng NumPy để sử dụng các phương pháp phân tích bên ngoài PyTorch, vốn được thực hiện trên CPU:

```
gnn_embeddings_np = gnn_embeddings.detach().cpu().numpy()
```

With this convention, we can produce UMAP calculations & visualization following process we used in N2V case. Resulting scatterplot (Fig. 2.8: Visualization of embeddings generated from passing a graph through a GNN.) is a 1st glimpse at clusters within our graph. Add different shadings based on each node's label (left-, right-, or neural-leaning) to see that similar leaning books are fairly well grouped, given that these embeddings were constructed from topology alone.
– Với quy ước này, chúng ta có thể tạo ra các phép tính UMAP & trực quan hóa theo quy trình đã sử dụng trong trường hợp N2V. Biểu đồ phân tán kết quả (Hình 2.8: Trực quan hóa các nhúng được tạo ra khi truyền đồ thị qua mạng GNN.) là cái nhìn đầu tiên về các cụm trong đồ thị của chúng ta. Hãy thêm các hiệu ứng tô bóng khác nhau dựa trên nhãn của mỗi nút (nghiêng trái, phải hoặc nghiêng nơ-ron) để thấy rằng các sách nghiêng tương tự được nhóm khá tốt, vì các nhúng này chỉ được xây dựng từ tôpô.
Next, discuss both how GNN embeddings are used & how they differ from those produced with N2V.
– Tiếp theo, thảo luận về cách sử dụng nhúng GNN & cách chúng khác với nhúng được tạo bằng N2V như thế nào.

* 2.2.2. GNN vs. N2V embeddings. Through this book, predominantly use GNNs to generate embeddings because this embedding process is intrinsic to a GNN's architecture. While embeddings play a pivotal role in methodologies & applications we explore in rest of book, their presence is often subtle & not always highlighted. This approach allows us to focus on broader concepts & applications of GNN-based ML without getting slowed down by technicalities. Nonetheless, important to acknowledge: underlying power & adaptability of embeddings are central to advanced techniques & insights we get into throughout text.
– GNN so với nhúng N2V. Trong cuốn sách này, chúng tôi chủ yếu sử dụng GNN để tạo nhúng vì quá trình nhúng này là 1 phần không thể thiếu trong kiến trúc của GNN. Mặc dù nhúng đóng vai trò then chốt trong các phương pháp luận & ứng dụng mà chúng tôi sẽ đề cập trong phần còn lại của cuốn sách, nhưng sự hiện diện của chúng thường rất tinh tế & không phải lúc nào cũng được nhấn mạnh. Cách tiếp cận này cho phép chúng tôi tập trung vào các khái niệm rộng hơn & ứng dụng của ML dựa trên GNN mà không bị chậm lại bởi các vấn đề kỹ thuật. Tuy nhiên, điều quan trọng cần lưu ý là: sức mạnh tiềm ẩn & khả năng thích ứng của nhúng là trọng tâm của các kỹ thuật tiên tiến & những hiểu biết sâu sắc mà chúng tôi sẽ tìm hiểu trong suốt cuốn sách.
GNN-produced node embeddings are particularly powerful because they enable us to tackle a broad range of graph-related tasks by using their inductive nature. Inductive learning allows these embeddings to generalize to new, unseen nodes or even entirely new graphs without needing to retrain model. In contrast, N2V embeddings are limited to specific graphs they were trained on & can't easily adapt to new data. Reiterate key ways in which GNN embeddings differ from other embedding methods, e.g. N2V [1, 3].
– Nhúng nút do GNN tạo ra đặc biệt mạnh mẽ vì chúng cho phép chúng ta giải quyết 1 loạt các tác vụ liên quan đến đồ thị bằng cách sử dụng tính chất quy nạp của chúng. Học quy nạp cho phép các nhúng này tổng quát hóa sang các nút mới, chưa từng thấy hoặc thậm chí là các đồ thị hoàn toàn mới mà không cần phải đào tạo lại mô hình. Ngược lại, nhúng N2V bị giới hạn trong các đồ thị cụ thể mà chúng được đào tạo & không thể dễ dàng thích ứng với dữ liệu mới. Nhắc lại

những điểm chính mà nhúng GNN khác với các phương pháp nhúng khác, e.g.: N2V [1, 3].

- Adaptability of new graphs. 1 of critical features of GNN embeddings is their adaptability. Because GNNs learn a function that maps node features to embeddings, this function can be applied to nodes in new graphs without needing to be retrained, provided nodes have similar feature spaces. This inductive capability is particularly valuable in dynamic environments where graph may evolve over time or in applications where model needs to be applied to different but structurally similar graphs. N2V, on other hand, needs to be reapplied for each new graph or set of nodes.
  – Khả năng thích ứng của đồ thị mới. 1 trong những đặc điểm quan trọng của nhúng GNN là khả năng thích ứng. Vì GNN học 1 hàm ánh xạ các đặc trưng của nút với các nhúng, hàm này có thể được áp dụng cho các nút trong đồ thị mới mà không cần phải đào tạo lại, miễn là các nút có không gian đặc trưng tương tự. Khả năng quy nạp này đặc biệt hữu ích trong các môi trường động, nơi đồ thị có thể phát triển theo thời gian hoặc trong các ứng dụng cần áp dụng mô hình cho các đồ thị khác nhau nhưng có cấu trúc tương tự. Mặt khác, N2V cần được áp dụng lại cho mỗi đồ thị hoặc tập hợp nút mới.
- Enhanced feature integration. GNNs inherently consider node features during embedding process, allowing for a complex & nuanced representation of each node. This integration of node features, alongside structural information, offers a more comprehensive view compared to N2V & other methods that focus on a graph's topology. This capability makes GNN embeddings particularly suited for tasks where node features contain significant additional information.
  – Tích hợp tính năng nâng cao. GNN vốn đã xem xét các tính năng nút trong quá trình nhúng, cho phép biểu diễn phức tạp & sắc thái của từng nút. Việc tích hợp các tính năng nút này, cùng với thông tin cấu trúc, mang lại cái nhìn toàn diện hơn so với các phương pháp N2V & khác tập trung vào cấu trúc của đồ thị. Khả năng này làm cho nhúng GNN đặc biệt phù hợp cho các tác vụ mà các tính năng nút chứa thông tin bổ sung đáng kể.
- Task-specific optimization. GNNs embeddings are trained alongside specific tasks, e.g. node classification, link prediction, or even graph classification. Through end-to-end training, GNN model learns to optimize embeddings for task at hand, leading to potentially higher performance & efficiency compared to using pre-generated embeddings e.g. those from N2V.
  – Tối ưu hóa theo tác vụ. Các nhúng GNN được huấn luyện cùng với các tác vụ cụ thể, e.g.: phân loại nút, dự đoán liên kết, hoặc thậm chí phân loại đồ thị. Thông qua quá trình huấn luyện đầu cuối, mô hình GNN học cách tối ưu hóa các nhúng cho tác vụ đang thực hiện, dẫn đến hiệu suất & hiệu quả cao hơn so với việc sử dụng các nhúng được tạo sẵn, e.g. từ N2V.

  That said, while GNN embeddings offer clear advantages in terms of adaptability & applicability to new data, N2V embeddings have their strengths, particularly in capturing nuanced patterns within a specific graph's structure. In practice, choice between GNN & N2V embeddings may depend on specific requirements of task, nature of graph data, & constraints of computational environment.
  – Tuy nhiên, trong khi nhúng GNN mang lại những lợi thế rõ ràng về khả năng thích ứng & khả năng ứng dụng cho dữ liệu mới, nhúng N2V cũng có những điểm mạnh riêng, đặc biệt là trong việc nắm bắt các mẫu hình phức tạp trong cấu trúc đồ thị cụ thể. Trên thực tế, việc lựa chọn giữa nhúng GNN & N2V có thể phụ thuộc vào các yêu cầu cụ thể của tác vụ, bản chất của dữ liệu đồ thị, & các ràng buộc của môi trường tính toán.

  For tasks where graph structure is static & well-defined, N2V might provide a simpler & computational efficient solution. Conversely, for dynamic graphs, large-scale applications, or scenarios requiring incorporation of node features, GNNs will often be the more robust & versatile choice. Additionally, when task itself is not well-defined & work is exploratory, N2V is likely faster & easier to use.
  – Đối với các tác vụ có cấu trúc đồ thị tĩnh & được xác định rõ ràng, N2V có thể cung cấp 1 giải pháp đơn giản hơn & hiệu quả về mặt tính toán. Ngược lại, đối với các đồ thị động, ứng dụng quy mô lớn hoặc các tình huống yêu cầu tích hợp các đặc điểm nút, GNN thường là lựa chọn mạnh mẽ & linh hoạt hơn. Ngoài ra, khi bản thân tác vụ không được xác định rõ & công việc mang tính khám phá, N2V có thể nhanh hơn & dễ sử dụng hơn.

  Have now successfully built our 1st GNN embedding. This is key 1st step for all GNN models, & everything from this point will build on it. In next sect, give an example of some of these next steps & show how to use embeddings to solve a ML problem.
  – Chúng ta đã xây dựng thành công mô hình nhúng GNN đầu tiên. Đây là bước đầu tiên quan trọng cho tất cả các mô hình GNN, & mọi thứ từ đây sẽ được xây dựng dựa trên nó. Trong phần tiếp theo, đưa ra ví dụ về 1 số bước tiếp theo & trình bày cách sử dụng nhúng để giải quyết bài toán học máy.

- 2.3. Using Node Embeddings. Semi-supervised learning, which involves a combination of labeled & unlabeled data, provides a valuable opportunity to compare different embedding techniques. In this chap, explore how GNN & N2V embeddings can be used to predict labels when majority of data lacks labels.
  – Sử dụng Nhúng Nút. Học bán giám sát, bao gồm sự kết hợp giữa dữ liệu có nhãn & không có nhãn, mang đến 1 cơ hội quý giá để so sánh các kỹ thuật nhúng khác nhau. Trong chương này, khám phá cách sử dụng nhúng GNN & N2V để dự đoán nhãn khi phần lớn dữ liệu thiếu nhãn.

  Our task involves Political Books dataset `books_graph`, where nodes represent political books & edges indicate co-purchase relationships. To make process clearer, review steps taken so far & outline our next steps, as illustrated in Fig. 2.9: Overview of steps taken in Chap. 2: 1. Preprocess Political Books dataset for embedding. 2. Use N2V & GCN to create embeddings from preprocessed data. 3. Prepare N2V embeddings & GCN embeddings for semi-supervised classification. 4. Embeddings are used as features in a random forest classifier (tabular features) & a GCN classifier (node features).
  – Nhiệm vụ của chúng tôi liên quan đến tập dữ liệu Sách Chính trị `books_graph`, trong đó các nút biểu diễn các cuốn sách chính trị & các cạnh biểu diễn mối quan hệ đồng mua. Để làm rõ quy trình, xem lại các bước đã thực hiện cho đến nay &

phác thảo các bước tiếp theo, như minh họa trong Hình 2.9: Tổng quan các bước đã thực hiện trong Chương 2: 1. Tiền xử lý tập dữ liệu Sách Chính trị để nhúng. 2. Sử dụng N2V & GCN để tạo nhúng từ dữ liệu đã được xử lý trước. 3. Chuẩn bị nhúng N2V & nhúng GCN cho phân loại bán giám sát. 4. Nhúng được sử dụng làm đặc trưng trong bộ phân loại rừng ngẫu nhiên (đặc trưng dạng bảng) & bộ phân loại GCN (đặc trưng nút).

Began with `books_graph` dataset in graph format & performed light preprocessing to prepare data for embedding. For N2V, this involved converting dataset from a `.gml` file to a `NetworkX` format. For GNN-based embeddings, converted `NetworkX` graph into a PyTorch tensor & initialized node features using Xavier initialization to ensure balanced variability across layers.

– Bắt đầu với tập dữ liệu `books_graph` ở định dạng đồ thị & thực hiện tiền xử lý nhẹ để chuẩn bị dữ liệu cho nhúng. Đối với N2V, việc này bao gồm chuyển đổi tập dữ liệu từ tệp `.gml` sang định dạng `NetworkX`. Đối với nhúng dựa trên GNN, chuyển đổi đồ thị `NetworkX` thành tensor PyTorch & khởi tạo các đặc trưng nút bằng khởi tạo Xavier để đảm bảo tính biến thiên cân bằng giữa các lớp.

After preparing data, we generated embeddings using both N2V & GCNs. Now, in this sect, apply these embeddings to a semi-supervised classification problem. This involves further processing to define classification task, where only 20% of book labels are retained, simulating a realistic scenario with sparse labeled data.

– Sau khi chuẩn bị dữ liệu, chúng tôi đã tạo ra các nhúng sử dụng cả N2V & GCN. Trong phần này, chúng tôi áp dụng các nhúng này vào 1 bài toán phân loại bán giám sát. Điều này bao gồm việc xử lý thêm để xác định tác vụ phân loại, trong đó chỉ giữ lại 20% nhãn sách, mô phỏng 1 kịch bản thực tế với dữ liệu được gắn nhãn thưa thớt.

Use 2 sets of embeddings (N2V & GCN) with 2 different classifiers: a random forest classifier (to use embeddings as tabular features) & a GCN classifier (to use graph structure & node features). Goal: predict political orientation of books, with remaining 80% of labels inferred based on given embeddings.

– Sử dụng 2 bộ nhúng (N2V & GCN) với 2 bộ phân loại khác nhau: 1 bộ phân loại rừng ngẫu nhiên (để sử dụng nhúng làm đặc trưng dạng bảng) & 1 bộ phân loại GCN (để sử dụng cấu trúc đồ thị & đặc trưng nút). Mục tiêu: dự đoán khuynh hướng chính trị của sách, với 80% nhãn còn lại được suy ra dựa trên các nhúng đã cho.

∗ 2.3.1. Data preprocessing. To start, we do a little more preprocessing to our `books_gml` dataset Listing 2.5: Preprocessing for semi-supervised problem. Must format labels in a suitable way for learning process. Because all nodes are labeled, we also have to set up semi-supervised problem by randomly selecting nodes from which we hide labels.

– Tiền xử lý dữ liệu. Để bắt đầu, chúng ta thực hiện thêm 1 chút tiền xử lý cho tập dữ liệu `books_gml` Liệt kê 2.5: Tiền xử lý cho bài toán bán giám sát. Phải định dạng nhãn theo cách phù hợp cho quá trình học. Vì tất cả các nút đều được gắn nhãn, chúng ta cũng phải thiết lập bài toán bán giám sát bằng cách chọn ngẫu nhiên các nút mà chúng ta ẩn nhãn. Nodes associated with attribute `'c'` are classified as `'right'`, while those with `'l'` are classified as `'left'`. Nodes that do not fit these criteria, including those with neutral or unspecified attributes, are categorized as `'neutral'`. These classifications are then placed into a NumPy array `labels` for optimized computational handling.

– Các nút liên kết với thuộc tính `'c'` được phân loại là `'right'`, trong khi các nút có thuộc tính `'l'` được phân loại là `'left'`. Các nút không phù hợp với các tiêu chí này, bao gồm cả các nút có thuộc tính trung lập hoặc không xác định, được phân loại là `'neutral'`. Các phân loại này sau đó được đưa vào mảng NumPy `nhãn` để tối ưu hóa xử lý tính toán. Then, an array `indices` is created, representing positional indexes of all nodes within dataset. A subset of these indices, corresponding to 20% of total node count, is designated as our labeled data.

– Sau đó, 1 mảng `indices` được tạo ra, biểu diễn các chỉ số vị trí của tất cả các nút trong tập dữ liệu. 1 tập hợp con của các chỉ số này, tương ứng với 20% tổng số nút, được chỉ định là dữ liệu được gắn nhãn của chúng tôi.

To manage labeled & unlabeled data, Boolean masks, `labelled_mask, unlabelled_mask`, are initialized & populated. `labelled_mask` is set to `True` for indices selected as labeled; these are ground truth labels for corresponding nodes. Similarly, `unlabelled_mask` is set to `False`. These masks segment dataset for training & evaluation, ensuring that algorithms are correctly trained & validated on correct subsets of data.

– Để quản lý dữ liệu được gắn nhãn & không gắn nhãn, các mặt nạ Boolean, `labelled_mask, unlabelled_mask`, được khởi tạo & điền. `labelled_mask` được đặt thành `True` cho các chỉ số được chọn là có gắn nhãn; đây là các nhãn thực tế cho các nút tương ứng. Tương tự, `unlabelled_mask` được đặt thành `False`. Các mặt nạ này phân đoạn tập dữ liệu để huấn luyện & đánh giá, đảm bảo rằng các thuật toán được huấn luyện & xác thực chính xác trên các tập con dữ liệu chính xác.

Listing 2.5: Preprocessing for semi-supervised problem.

Now transform data for model training, as shown in Listing 2.6: Preprocessing: constructing training data. For GNN-derived embeddings, `x_train_gnn, y_train_gnn` are assigned arrays of embeddings & corresponding numeric labels filtered by a `labelled_mask`. This mask is a Boolean array indicating which nodes in graph are part of labeled subset, ensuring that only data points with known labels are included in training set.

– Bây giờ, hãy biến đổi dữ liệu để huấn luyện mô hình, như được hiển thị trong Liệt kê 2.6: Tiền xử lý: xây dựng dữ liệu huấn luyện. Đối với các nhúng được lấy từ GNN, `x_train_gnn, y_train_gnn` được gán các mảng nhúng & nhãn số tương ứng, được lọc bởi `labelled_mask`. Mặt nạ này là 1 mảng Boolean cho biết các nút nào trong đồ thị là 1 phần của tập con được gắn nhãn, đảm bảo rằng chỉ những điểm dữ liệu có nhãn đã biết mới được đưa vào tập huấn luyện.

For N2V embeddings, a similar approach is adopted with an added preprocessing step to align embeddings with their corresponding labels. Embeddings for each node aggregated into NumPy array `X_n2v` in same order as nodes appear in `books_graph`. This ensures consistency between embeddings & their labels, a crucial step for supervised learning tasks. Subsequently, `X_train_n2v, y_train_n2v` are populated with N2V embeddings & labels, again applying `labelled_mask` to filter for labeled data points.

– Đối với nhúng N2V, 1 phương pháp tương tự được áp dụng với bước tiền xử lý bổ sung để căn chỉnh các nhúng với nhãn tương ứng. Các nhúng cho mỗi nút được tổng hợp thành mảng NumPy `X_n2v` theo cùng thứ tự các nút xuất hiện trong `books_graph`. Điều này đảm bảo tính nhất quán giữa các nhúng & nhãn của chúng, 1 bước quan trọng cho các tác vụ học có giám sát. Sau đó, `X_train_n2v, y_train_n2v` được điền các nhúng N2V & nhãn, 1 lần nữa áp dụng `labelled_mask` để lọc các điểm dữ liệu được gắn nhãn.

```
# preprocessing: constructing training data
## for GNN embeddings
X_train_gnn = gnn_embeddings[labelled_mask]
y_train_gnn = numeric_labels[labelled_mask]

X_n2v = np.array([embeddings[str(node)] for node in gml_graph.nodes()]) # for N2V embeddings
## ensure N2V embeddings are in the same order as labels
X_train_n2v = X_n2v[labelled_mask]
y_train_n2v = numeric_labels[labelled_mask]
```

Extra alignment step for N2V embeddings is not necessary for GNN embeddings because GNN models inherently maintain order of nodes as they process entire graph in a structured manner. As a result, output embeddings from a GNN are naturally ordered in correspondence with input graph's node order.

– Bước căn chỉnh bổ sung cho nhúng N2V không cần thiết cho nhúng GNN vì các mô hình GNN vốn dĩ duy trì thứ tự các nút khi chúng xử lý toàn bộ đồ thị theo cách có cấu trúc. Do đó, các nhúng đầu ra từ GNN được sắp xếp tự nhiên theo thứ tự các nút của đồ thị đầu vào.

In contrast, N2V generates embeddings through independent random walks starting from each node, & order of resulting embeddings does not necessarily match order of nodes in original graph data structure. Therefore, an explicit alignment step is required to ensure that each N2V embedding is correctly associated with its corresponding label, as extracted from graph. This step is critical for supervised learning tasks where correct matching of features (embeddings) to labels is essential for model training & evaluation. For this task, use attribute `index_to_key`, which contains identifiers of nodes in order they are processed & stored within model.

– Ngược lại, N2V tạo ra các nhúng thông qua các bước ngẫu nhiên độc lập bắt đầu từ mỗi nút, & thứ tự của các nhúng kết quả không nhất thiết phải khớp với thứ tự của các nút trong cấu trúc dữ liệu đồ thị gốc. Do đó, cần có 1 bước căn chỉnh rõ ràng để đảm bảo mỗi nhúng N2V được liên kết chính xác với nhãn tương ứng của nó, được trích xuất từ đồ thị. Bước này rất quan trọng đối với các tác vụ học có giám sát, trong đó việc khớp chính xác các đặc trưng (nhúng) với nhãn là điều cần thiết cho việc huấn luyện mô hình & đánh giá. Đối với tác vụ này, hãy sử dụng thuộc tính `index_to_key`, chứa các định danh của các nút theo thứ tự chúng được xử lý & lưu trữ trong mô hình.

  - ∗ 2.3.2. Random forest classification. With our data prepped, use GNN & N2V embeddings from Sects. 2.1–2.2 as input features for a `RandomForestClassifier`, as shown in Listing 2.7: Preprocessing: constructing training data. p. 52+++
  - ○ 2.4. Under Hood.

PART 3: GRAPH NEURAL NETWORKS GNNs

- 3. Graph convolutional networks & GraphSAGE.
- 4. Graph attention networks.
- 5. Graph autoencoders.

PART 3: ADVANCED TOPICS.

- 6. Dynamic graphs: Spatiotemporal GNNs.
- 7. Learning & inference at scale.
- 8. Considerations for GNN projects.
- A. Discovering graphs.
- B. Installing & configuring PyTorch Geometric.

1.2 Maxime Labonne. Hands-On Graph Neural Networks Using Python: Practical Techniques & Architectures for Building Powerful Graph & DL Apps with PyTorch. 2023

- Preface. In just 10 years, GNNs have become an essential & popular DL architecture. They have already had a significant impact various industries, e.g. in drug discovery, where GNNs predicted a new antibiotic, named halicin, & have improved estimated time of arrival calculations on Google Maps. Tech companies & universities are exploring potential of GNNs in various applications, including recommender systems, fake news detection, & chip design. GNNs have enormous potential & many yet-to-be-discovered applications, making them a critical tool for solving global problems.

  – Chỉ trong 10 năm, GNN đã trở thành 1 kiến trúc DL thiết yếu & phổ biến. Chúng đã có tác động đáng kể đến nhiều ngành công nghiệp, e.g. trong lĩnh vực khám phá thuốc, nơi GNN dự đoán 1 loại kháng sinh mới, tên là halicin, & đã cải thiện khả

năng tính toán thời gian đến ước tính trên Google Maps. Các công ty công nghệ & các trường đại học đang khám phá tiềm năng của GNN trong nhiều ứng dụng khác nhau, bao gồm hệ thống đề xuất, phát hiện tin giả, & thiết kế chip. GNN có tiềm năng to lớn & nhiều ứng dụng chưa được khám phá, khiến chúng trở thành 1 công cụ quan trọng để giải quyết các vấn đề toàn cầu.

In this book, aim to provide a comprehensive & practical overview of world of GNNs. Begin by exploring fundamental concepts of graph theory & graph learning & then delve into most widely used & well-established GNN architectures. As we progress, also cover latest advances in GNNs & introduce specialized architectures that are designed to tackle specific tasks, e.g. graph generation, link prediction, & more.

– Trong cuốn sách này, chúng tôi đặt mục tiêu cung cấp 1 cái nhìn tổng quan toàn diện & thực tiễn về thế giới GNN. Bắt đầu bằng việc khám phá các khái niệm cơ bản của lý thuyết đồ thị & học đồ thị & sau đó đi sâu vào các kiến trúc GNN được sử dụng rộng rãi nhất & đã được thiết lập tốt. Khi chúng ta tiến triển, chúng tôi cũng sẽ đề cập đến những tiến bộ mới nhất trong GNN & giới thiệu các kiến trúc chuyên biệt được thiết kế để giải quyết các nhiệm vụ cụ thể, e.g.: tạo đồ thị, dự đoán liên kết, & nhiều hơn nữa.

In addition to these specialized chaps, provide hands-on experience through 3 practical projects. These projects will cover critical real-world applications of GNNs, including traffic forecasting, anomaly detection, & recommender systems. Through these projects, gain a deeper understanding of how GNNs work & also develop skills to implement them in practical scenarios.

– Ngoài các chuyên gia này, chương trình còn cung cấp kinh nghiệm thực tế thông qua 3 dự án thực tế. Các dự án này sẽ đề cập đến các ứng dụng quan trọng của GNN trong thế giới thực, bao gồm dự báo lưu lượng truy cập, phát hiện bất thường & hệ thống đề xuất. Thông qua các dự án này, bạn sẽ hiểu sâu hơn về cách thức hoạt động của GNN & phát triển kỹ năng triển khai chúng trong các tình huống thực tế.

Finally, this book provides a hands-on learning experience with readable code for every chap's techniques & relevant applications, which are readily accessible on GitHub & Google Colab. By end of this book, have a comprehensive understanding of field of graph learning & GNNs & will be well-equipped to design & implement these models for a wide range of applications.

– Cuối cùng, cuốn sách này cung cấp trải nghiệm học tập thực hành với mã nguồn dễ đọc cho mọi kỹ thuật & các ứng dụng liên quan, có thể dễ dàng truy cập trên GitHub & Google Colab. Khi hoàn thành cuốn sách này, bạn sẽ có hiểu biết toàn diện về lĩnh vực học đồ thị & Mạng lưới Mạng Thần kinh Nhân tạo (GNN) & được trang bị tốt để thiết kế & triển khai các mô hình này cho nhiều ứng dụng khác nhau.

- Who this book is for. This book is intended for individuals interested in learning about GNNs & how they can be applied to various real-world problems. This book is ideal for *data scientists, ML engineers, & AI* professionals who want to gain practical experience in designing & implementing GNNs. This book is written for individuals with prior knowledge of DL & ML. However, it provides a comprehensive introduction to fundamental concepts of graph theory & graph learning for those new to field. Also be useful for researchers & students in computer science, mathematics, & engineering who want to expand their knowledge in this rapidly growing area of research.

  –Cuốn sách này dành cho những cá nhân quan tâm đến việc tìm hiểu về mạng lưới thần kinh nhân tạo (GNN) & cách chúng có thể được áp dụng cho các vấn đề thực tế khác nhau. Cuốn sách này lý tưởng cho các nhà khoa học dữ liệu, kỹ sư ML, & AI chuyên gia muốn tích lũy kinh nghiệm thực tế trong việc thiết kế & triển khai GNN. Cuốn sách này được viết cho những cá nhân đã có kiến thức về DL & ML. Tuy nhiên, nó cung cấp 1 giới thiệu toàn diện về các khái niệm cơ bản của lý thuyết đồ thị & học đồ thị cho những người mới bắt đầu. Nó cũng hữu ích cho các nhà nghiên cứu & sinh viên ngành khoa học máy tính, toán học, & kỹ thuật muốn mở rộng kiến thức trong lĩnh vực nghiên cứu đang phát triển nhanh chóng này.

- What this book covers. Chap. 1: Getting Started with Graph Learning, provides a comprehensive introduction to GNNs, including their importance in modern data analysis & ML. Chap starts by exploring relevance of graphs as a representation of data & their widespread use in various domains. It then delves into importance of graph learning, including different applications & techniques. Finally, chap focuses on GNN architecture & highlights its unique features & performance compared to other methods.

  – Chương 1: Bắt đầu với Học đồ thị, cung cấp phần giới thiệu toàn diện về GNN, bao gồm tầm quan trọng của chúng trong phân tích dữ liệu hiện đại & Học máy (ML). Chương này bắt đầu bằng việc khám phá tính liên quan của đồ thị như 1 phương tiện biểu diễn dữ liệu & ứng dụng rộng rãi của chúng trong nhiều lĩnh vực. Sau đó, chương đi sâu vào tầm quan trọng của học đồ thị, bao gồm các ứng dụng & kỹ thuật khác nhau. Cuối cùng, chương tập trung vào kiến trúc GNN & làm nổi bật các tính năng độc đáo & hiệu suất của nó so với các phương pháp khác.

  Chap. 2: Graph Theory for Graph Neural Networks, covers basics of graph theory & introduces various types of graphs, including their properties & applications. This chap also covers fundamental graph concepts, e.g. adjacency matrix, graph measures, e.g. centrality, & graph algorithms, BFS & DFS.

  – Chương 2: Lý thuyết Đồ thị cho Mạng Nơ-ron Đồ thị, bao gồm những kiến thức cơ bản về lý thuyết đồ thị & giới thiệu các loại đồ thị khác nhau, bao gồm các tính chất & ứng dụng của chúng. Chương này cũng đề cập đến các khái niệm cơ bản về đồ thị, e.g. ma trận kề, độ đo đồ thị, e.g. độ tập trung, & thuật toán đồ thị, BFS & DFS.

  Chap. 3: Creating Node Representations with DeepWalk, focused on DeepWalk, a pioneer in applying ML to graph data. Main objective of DeepWalk architecture: generate node representations that other models can utilize for downstream tasks e.g. node classification. Chap covers 2 key components of DeepWalk – Word2Vec & random walks – with a particular emphasis on Word2Vec skip-gram model.

– Chương 3: Tạo Biểu diễn Nút với DeepWalk, tập trung vào DeepWalk, 1 công nghệ tiên phong trong việc áp dụng Học máy (ML) vào dữ liệu đồ thị. Mục tiêu chính của kiến trúc DeepWalk: tạo ra các biểu diễn nút mà các mô hình khác có thể sử dụng cho các tác vụ hạ nguồn, e.g. phân loại nút. Chương này đề cập đến 2 thành phần chính của DeepWalk – Word2Vec & random walks – đặc biệt nhấn mạnh vào mô hình Word2Vec skip-gram.

Chap. 4: Improving Embeddings with Biased Random Walks in Node2Vec, focused on Node2Vec architecture, which is based on DeepWalk architecture covered in previous chap. Chap covers modifications made to random walk generation in Node2Vec & how to select best parameters for a specific graph. Implementation of Node2Vec is compared to DeepWalk on Zachary's Karate Club to highlight differences between 2 architectures. Chap concludes with a practical application of Node2Vec, building a movie recommendation system.

– Chương 4: Cải thiện nhúng với bước ngẫu nhiên có thiên vị trong Node2Vec, tập trung vào kiến trúc Node2Vec, dựa trên kiến trúc DeepWalk đã được đề cập trong chương trước. Chương này đề cập đến những thay đổi được thực hiện đối với việc tạo bước ngẫu nhiên trong Node2Vec & cách chọn tham số tốt nhất cho 1 đồ thị cụ thể. Việc triển khai Node2Vec được so sánh với DeepWalk trên Zachary's Karate Club để làm nổi bật sự khác biệt giữa 2 kiến trúc. Chương kết thúc bằng 1 ứng dụng thực tế của Node2Vec, xây dựng hệ thống đề xuất phim.

Chap. 5: Including Node Features with Vanilla Neural Networks, explores integration of additional information, e.g. node & edge features, into graph embeddings to produce more accurate results. Chap starts with a comparison of vanilla neural network's performance on node features only, treated as tabular datasets. Then, experiment with adding topological information to neural networks, leading to creation of a simple vanilla GNN architecture.

– Chương 5: Bao gồm các đặc điểm nút với mạng nơ-ron nhân tạo thuần túy, khám phá việc tích hợp thông tin bổ sung, e.g. đặc điểm nút & cạnh, vào nhúng đồ thị để tạo ra kết quả chính xác hơn. Chương này bắt đầu bằng việc so sánh hiệu suất của mạng nơ-ron nhân tạo thuần túy chỉ trên các đặc điểm nút, được xử lý dưới dạng tập dữ liệu bảng. Sau đó, thử nghiệm thêm thông tin tôpô vào mạng nơ-ron, dẫn đến việc tạo ra 1 kiến trúc mạng nơ-ron nhân tạo thuần túy đơn giản.

Chap. 6: Introducing Graph Convolutional Networks, focuses on Graph Convolutional Network (GCN) architecture & its importance as a blueprint for GNNs. It covers limitations of previous vanilla GNN layers & explains motivation behind GCNs. Chap details how GCN layer works, its performance improvements over vanilla GNN layer, & its implementation on Cora & Facebook Page-Page datasets using PyTorch Geometric. Chap also touches upon task of node regression & benefits of transforming tabular data into a graph.

– Chương 6: Giới thiệu về Mạng Tích chập Đồ thị, tập trung vào kiến trúc Mạng Tích chập Đồ thị (GCN) & tầm quan trọng của nó như 1 bản thiết kế cho GNN. Chương này đề cập đến những hạn chế của các lớp GNN thuần túy trước đây & giải thích động lực đằng sau GCN. Chương này trình bày chi tiết cách thức hoạt động của lớp GCN, những cải tiến về hiệu suất so với lớp GNN thuần túy, & triển khai nó trên Cora & bộ dữ liệu Facebook Page-Page bằng PyTorch Geometric. Chương cũng đề cập đến nhiệm vụ của hồi quy nút & lợi ích của việc chuyển đổi dữ liệu dạng bảng thành dạng đồ thị.

Chap. 7: Graph Attention Networks, focuses on Graph Attention Networks (GATs), which are an improvements over GCNs. Chap explains how GATs work by using concepts of self-attention & provides a step-by-step understanding of graph attention layer. Chap also implements a graph attention layer from scratch using NumPy. Final section of chap discusses use of a GAT on 2 node classification datasets, Cora & CiteSeer, & compares accuracy with that of a GCN.

– Chương 7: Mạng lưới Chú ý Đồ thị, tập trung vào Mạng lưới Chú ý Đồ thị (GAT), 1 cải tiến so với GCN. Chương này giải thích cách thức hoạt động của GAT bằng cách sử dụng các khái niệm tự chú ý & cung cấp hiểu biết từng bước về lớp chú ý đồ thị. Chương cũng triển khai 1 lớp chú ý đồ thị từ đầu bằng NumPy. Phần cuối của chương thảo luận về việc sử dụng GAT trên các tập dữ liệu phân loại 2 nút, Cora & CiteSeer, & so sánh độ chính xác với GCN.

Chap. 8: Scaling up GNNs with GraphSAGE, focuses on GraphSAGE architecture & its ability to handle large graphs effectively. Chap covers 2 main ideas behind GraphSAGE, including its neighbor sampling technique & aggregation operators. Learn about variants proposed by tech companies e.g. Uber Eats & Pinterest, as well as benefits of GraphSAGE's inductive approach. Chap concludes by implementing GraphSAGE for node classification & multi-label classification tasks.

– Chương 8: Mở rộng mạng lưới mô hình dữ liệu lớn (GNN) với GraphSAGE, tập trung vào kiến trúc GraphSAGE & khả năng xử lý đồ thị lớn hiệu quả. Chương này đề cập đến 2 ý tưởng chính đằng sau GraphSAGE, bao gồm kỹ thuật lấy mẫu lân cận & các toán tử tổng hợp. Tìm hiểu về các biến thể được đề xuất bởi các công ty công nghệ, e.g. Uber Eats & Pinterest, cũng như lợi ích của phương pháp quy nạp của GraphSAGE. Chương kết thúc bằng việc triển khai GraphSAGE cho các tác vụ phân loại nút & phân loại đa nhãn.

Chap. 9: Defining Expressiveness for Graph Classification, explores concept of expressiveness in GNNs & how it can be used to design better models. It introduces Weisfeiler-Leman (WL) test, which provides framework for understanding expressiveness in GNNs. Chap uses WL test to compare different GNN layers & determine most expressive one. Based on this result, a more powerful GNN is designed & implemented by PyTorch Geometric. Chap concludes with a comparison of different methods for graph classification on PROTEINs dataset.

– Chương 9: Định nghĩa tính biểu cảm cho phân loại đồ thị, khám phá khái niệm tính biểu cảm trong mạng lưới biểu diễn dữ liệu (GNN) & cách sử dụng nó để thiết kế các mô hình tốt hơn. Chương này giới thiệu kiểm định Weisfeiler-Leman (WL), cung cấp khuôn khổ để hiểu tính biểu cảm trong GNN. Chương sử dụng kiểm định WL để so sánh các lớp GNN khác nhau & xác định lớp biểu cảm nhất. Dựa trên kết quả này, 1 GNN mạnh mẽ hơn đã được thiết kế & triển khai bởi PyTorch Geometric. Chương kết thúc bằng việc so sánh các phương pháp phân loại đồ thị khác nhau trên tập dữ liệu PROTEIN.

Chap. 10: Predicting Links with GNNs, focuses on link prediction in graphs. It covers traditional techniques, e.g. matrix factorization & GNN-based methods. Chap explains concept of link prediction & its importance in social networks & recommender systems. Learn about limitations of traditional techniques & benefits of using GNN-based methods. Exploree

3 GNN-based techniques from 2 different families, including node embeddings & subgraph representation. Finally, implement various link prediction techniques in PyTorch Geometric & choose best method for a given problem.

– Chương 10: Dự đoán Liên kết với GNN, tập trung vào dự đoán liên kết trong đồ thị. Chương này đề cập đến các kỹ thuật truyền thống, e.g. phân tích ma trận & các phương pháp dựa trên GNN. Chương này giải thích khái niệm dự đoán liên kết & tầm quan trọng của nó trong mạng xã hội & các hệ thống đề xuất. Tìm hiểu về những hạn chế của các kỹ thuật truyền thống & lợi ích của việc sử dụng các phương pháp dựa trên GNN. Khám phá 3 kỹ thuật dựa trên GNN từ 2 họ khác nhau, bao gồm nhúng nút & biểu diễn đồ thị con. Cuối cùng, triển khai các kỹ thuật dự đoán liên kết khác nhau trong PyTorch Geometric & chọn phương pháp tốt nhất cho 1 bài toán nhất định.

Chap. 11: Generating Graphs Using GNNs, explores field of graph generation, which involves finding methods to create new graphs. Chap 1st introduces you to traditional techniques e.g. Erdös–Rényi & small-world models. Then focus on 3 families of solutions for GNN-based graph generation: VAE-based, autoregressive, & GAN-based models. Chap concludes with an implementation of a GAN-based framework with Reinforcement Learning (RL) to generate new chemical compounds using DeepChem library with TensorFlow.

– Chương 11: Tạo đồ thị bằng GNN, khám phá lĩnh vực tạo đồ thị, bao gồm việc tìm kiếm các phương pháp để tạo đồ thị mới. Chương 1 giới thiệu các kỹ thuật truyền thống, e.g. mô hình Erdos–Renyi & mô hình thế giới nhỏ. Sau đó, tập trung vào 3 nhóm giải pháp tạo đồ thị dựa trên GNN: mô hình dựa trên VAE, mô hình tự hồi quy & mô hình GAN. Chương kết thúc bằng việc triển khai 1 khuôn khổ dựa trên GAN với Học Tăng cường (RL) để tạo ra các hợp chất hóa học mới bằng thư viện DeepChem với TensorFlow.

Chap. 12: Learning from Heterogeneous Graphs, focuses on heterogeneous GNNs. Heterogeneous graphs contain different types of nodes & edges, in contrast in homogeneous graphs, which only involve 1 type of node & 1 type of edge. Chap begins by reviewing *Message Passing Neural Network* (MPNN) framework for homogeneous GNNs, then expands framework to heterogeneous networks. Finally, introduce a technique for creating a heterogeneous dataset, transforming homogeneous architectures into heterogeneous ones, & discussing an architecture specifically designed for processing heterogeneous networks.

– Chương 12: Học từ Đồ thị Không đồng nhất, tập trung vào Mạng nơ-ron nhân tạo không đồng nhất (GNN) không đồng nhất. Đồ thị không đồng nhất chứa các loại nút & cạnh khác nhau, trái ngược với đồ thị đồng nhất, chỉ bao gồm 1 loại nút & 1 loại cạnh. Chương bắt đầu bằng việc xem xét khuôn khổ Mạng nơ-ron truyền thông điệp (MPNN) cho GNN đồng nhất, sau đó mở rộng khuôn khổ sang các mạng không đồng nhất. Cuối cùng, giới thiệu 1 kỹ thuật để tạo 1 tập dữ liệu không đồng nhất, chuyển đổi các kiến trúc đồng nhất thành kiến trúc không đồng nhất, & thảo luận về 1 kiến trúc được thiết kế riêng để xử lý các mạng không đồng nhất.

Chap. 13: Temporal GNNs, focuses on Temporal GNNs, or Spatio-Temporal GNNs, which are a type of GNN that can handle graphs with changing edges & features over time. Chap 1st explains concept of dynamic graphs & applications of temporal GNNs, focusing on time series forecasting. Chap then moves on to application of temporal GNNs to web traffic forecasting to improve results using temporal information. Finally, chap describes another temporal GNN architecture specifically designed for dynamic graphs & applies it to task of epidemic forecasting.

– Chương 13: Mạng GNN thời gian, tập trung vào Mạng GNN thời gian, hay Mạng GNN không gian-thời gian, là 1 loại Mạng GNN có thể xử lý đồ thị với các cạnh & đặc trưng thay đổi theo thời gian. Chương 1 giải thích khái niệm về đồ thị động & ứng dụng của Mạng GNN thời gian, tập trung vào dự báo chuỗi thời gian. Sau đó, chương chuyển sang ứng dụng Mạng GNN thời gian vào dự báo lưu lượng truy cập web để cải thiện kết quả bằng cách sử dụng thông tin thời gian. Cuối cùng, chương này mô tả 1 kiến trúc Mạng GNN thời gian khác được thiết kế riêng cho đồ thị động & ứng dụng nó vào nhiệm vụ dự báo dịch bệnh.

- Chap. 14: Explaining GNNs, covers various techniques to better understands predictions & behavior of a GNN model. Chap highlights 2 popular explanation methods: GNNExplainer & integrated gradients. Then, see application of these techniques on a graph classification task using MUTAG dataset & a node classification task using Twitch social network.

  – Chương 14: Giải thích về mạng GNN, bao gồm các kỹ thuật khác nhau để hiểu rõ hơn về dự đoán & hành vi của mô hình GNN. Chương này nêu bật 2 phương pháp giải thích phổ biến: GNNExplainer & gradient tích hợp. Sau đó, xem xét ứng dụng của các kỹ thuật này trên 1 bài toán phân loại đồ thị sử dụng tập dữ liệu MUTAG & 1 bài toán phân loại nút sử dụng mạng xã hội Twitch.

- Chap. 15: Forecasting Traffic Using A3T-GCN, focuses on application of Temporal Graph Neural Networks in field of traffic forecasting. It highlights importance of accurate traffic forecasts in smart cities & challenges of traffic forecasting due to complex spatial & temporal dependencies. Chap covers steps involved in processing a new dataset to create a temporal graph & implementation of a new type of temporal GNN to predict future traffic speed. Finally, results are compared to a baseline solution to verity relevance of architecture.

  – Chương 15: Dự báo lưu lượng giao thông bằng A3T-GCN, tập trung vào ứng dụng của Mạng nơ-ron đồ thị thời gian trong lĩnh vực dự báo giao thông. Bài viết nhấn mạnh tầm quan trọng của việc dự báo giao thông chính xác trong các thành phố thông minh & những thách thức của việc dự báo giao thông do sự phụ thuộc phức tạp về không gian & thời gian. Chương này trình bày các bước xử lý tập dữ liệu mới để tạo đồ thị thời gian & triển khai 1 loại Mạng nơ-ron đồ thị thời gian mới để dự đoán tốc độ giao thông trong tương lai. Cuối cùng, kết quả được so sánh với giải pháp cơ sở để đánh giá tính phù hợp thực tế của kiến trúc.

- Chap. 16: Recommending Books Using LightGCN, focuses on application of GNNs in recommender systems. Goal of recommender systems: provide personalized recommendations to users based on their interests & past interactions. GNNs are

well-suited for this task as they can effectively incorporate complex relationships between users & items. In this chap, LightGCN architecture is introduced as a GNN specifically designed for recommender systems. Using Book-Crossing dataset, chap demonstrates how to build a book recommender system with collaborative filtering using LightGCN architecture.

– Chương 16: Đề xuất sách bằng LightGCN, tập trung vào ứng dụng của GNN trong hệ thống đề xuất. Mục tiêu của hệ thống đề xuất: cung cấp các đề xuất được cá nhân hóa cho người dùng dựa trên sở thích & các tương tác trước đây của họ. GNN rất phù hợp cho nhiệm vụ này vì chúng có thể kết hợp hiệu quả các mối quan hệ phức tạp giữa người dùng & các mục. Trong chương này, kiến trúc LightGCN được giới thiệu như 1 GNN được thiết kế riêng cho hệ thống đề xuất. Sử dụng tập dữ liệu Book-Crossing, chương này trình bày cách xây dựng 1 hệ thống đề xuất sách với lọc cộng tác bằng kiến trúc LightGCN.

- Chap. 17: Recommending Books Using LightGCN, focuses on application of GNNs in recommender systems. Goal of recommender systems: provide personalized recommendations to users based on their interests & past interactions. GNNs are well-suited for this tasks as they can effectively incorporate complex relationships between users & items. In this chap, LightGCN architecture is introduced as a GNN specifically designed for recommender systems. Using Book-Crossing dataset, chap demonstrates how to build a book recommender system with collaborative filtering using LightGCN architecture.

  – Chương 17: Đề xuất sách bằng LightGCN, tập trung vào ứng dụng của GNN trong các hệ thống đề xuất. Mục tiêu của hệ thống đề xuất: cung cấp các đề xuất được cá nhân hóa cho người dùng dựa trên sở thích & các tương tác trước đây của họ. GNN rất phù hợp cho nhiệm vụ này vì chúng có thể kết hợp hiệu quả các mối quan hệ phức tạp giữa người dùng & các mục. Trong chương này, kiến trúc LightGCN được giới thiệu như 1 GNN được thiết kế riêng cho các hệ thống đề xuất. Sử dụng tập dữ liệu Book-Crossing, chương này trình bày cách xây dựng 1 hệ thống đề xuất sách với lọc cộng tác bằng kiến trúc LightGCN.

- Chap. 18: Unlocking Potential of GNNs for Real-World Applications, summarizes what we have learned throughout book, & looks ahead to future of GNNs.

  – Chương 18: Khai phá tiềm năng của GNN cho các ứng dụng thực tế, tóm tắt những gì chúng ta đã học được trong suốt cuốn sách, & hướng tới tương lai của GNN.

- To get most out of this book. Should have a basic understanding of graph theory & ML concepts, e.g. supervised & unsupervised learning, training, & evaluation of models to maximize your learning experience. Familiarity with DL frameworks, e.g. PyTorch, will also be useful, although not essential, as book will provide a comprehensive introduction to mathematical concepts & their implementation.

  – Để tận dụng tối đa cuốn sách này. Bạn nên có hiểu biết cơ bản về lý thuyết đồ thị & các khái niệm ML, e.g.: học có giám sát & học không giám sát, đào tạo, & đánh giá các mô hình để tối đa hóa trải nghiệm học tập của mình. Việc quen thuộc với các nền tảng DL, e.g.: PyTorch, cũng sẽ hữu ích, mặc dù không bắt buộc, vì cuốn sách sẽ cung cấp phần giới thiệu toàn diện về các khái niệm toán học & cách triển khai chúng.

  Software covered in book: Python 3.8.15, PyTorch 1.13.1, PyTorch Geometric 2.2.0.

PART 1: INTRODUCTION TO GRAPH LEARNING.

In recent years, graph representation of data has become increasingly prevalent across various domains, from social networks to molecular biology. It is crucial to have a deep understanding of GNNs, which are designed specifically to handle graph-structured data, to unlock full potential of this representation.

– Trong những năm gần đây, biểu diễn dữ liệu dạng đồ thị ngày càng trở nên phổ biến trong nhiều lĩnh vực, từ mạng xã hội đến sinh học phân tử. Việc hiểu sâu sắc về mạng lưới mô hình dữ liệu (GNN), được thiết kế chuyên biệt để xử lý dữ liệu có cấu trúc đồ thị, là rất quan trọng để khai thác toàn bộ tiềm năng của biểu diễn này.

This 1st part part consists of 2 chaps & serves as a solid foundation for rest of book. It introduces concepts of graph learning & GNNs & their relevance in numerous tasks & industries. It also covers fundamental concepts of graph theory & its applications in graph learning, e.g. graph centrality measures. This part also highlights unique features & performance of GNN architecture compared to other methods.

– Phần 1 này bao gồm 2 chương & đóng vai trò là nền tảng vững chắc cho phần còn lại của cuốn sách. Phần này giới thiệu các khái niệm về học đồ thị & Mạng lưới Mạng Toàn cầu (GNN) & sự liên quan của chúng trong nhiều tác vụ & ngành công nghiệp. Phần này cũng đề cập đến các khái niệm cơ bản của lý thuyết đồ thị & ứng dụng của nó trong học đồ thị, e.g. các phép đo tính trung tâm của đồ thị. Phần này cũng làm nổi bật các tính năng độc đáo & hiệu suất của kiến trúc GNN so với các phương pháp khác.

By end of this part, have a solid understanding of importance of GNNs in solving many real-world problems. Will be acquainted with essentials of graph learning & how it is used in various domains. Furthermore, will have a comprehensive overview of main concepts of graph theory used in later chaps. With this solid foundation, will be well equipped to move on to more advanced concepts in graph learning & GNNs in following parts of book.

– Đến cuối phần này, bạn sẽ hiểu rõ tầm quan trọng của mạng GNN trong việc giải quyết nhiều vấn đề thực tế. Bạn sẽ được làm quen với những kiến thức cơ bản về học đồ thị & cách thức ứng dụng nó trong nhiều lĩnh vực khác nhau. Hơn nữa, bạn sẽ có cái nhìn tổng quan toàn diện về các khái niệm chính của lý thuyết đồ thị được sử dụng trong các chương sau. Với nền tảng vững chắc này, bạn sẽ được trang bị tốt để tiếp tục học các khái niệm nâng cao hơn về học đồ thị & mạng GNN trong các phần tiếp theo của cuốn sách.

- 1. Getting Started with Graph Learning. In this chap, delve into foundations of GNNs & understand why they are crucial tools in modern data analysis & ML. To that end, answer 3 essential questions providing us with a comprehensive understanding of GNNs.

  – Trong chương này, chúng ta sẽ đi sâu vào nền tảng của GNN & hiểu tại sao chúng là công cụ quan trọng trong phân tích dữ liệu hiện đại & ML. Để đạt được mục tiêu đó, trả lời 3 câu hỏi thiết yếu, giúp chúng ta hiểu toàn diện về GNN.

  1st, explore significance of graphs as a representation of data, & why they are widely used in various domains e.g. CS, biology, & finance. Next, delve into importance of graph learning, where we will understand different applications of graph learning & different families of graph learning techniques. Finally, focus on GNN family, highlighting its unique features, performance, & how it stands out compared to other methods.

  – Đầu tiên, khám phá ý nghĩa của đồ thị trong việc biểu diễn dữ liệu, & lý do tại sao chúng được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, e.g. Khoa học Máy tính, Sinh học, & Tài chính. Tiếp theo, đi sâu vào tầm quan trọng của học đồ thị, nơi chúng ta sẽ tìm hiểu các ứng dụng khác nhau của học đồ thị & các nhóm kỹ thuật học đồ thị khác nhau. Cuối cùng, tập trung vào họ GNN, làm nổi bật các tính năng độc đáo, hiệu suất của nó, & cách nó nổi bật so với các phương pháp khác.

  By end of this chap, have a clear understanding of why GNns are important & how they can be used to solve real-world problems. Will also be equipped with knowledge & skills you need to dive deeper into more advanced topics.

  - 1.1. Why graphs? 1st question we need to address: why interested in graphs in 1st place? Graph theory, mathematical study of graphs, has emerged as a fundamental tool for understanding complex systems & relationships. A graph is a visual representation of a collection of nodes (also called vertices) & edges that connect these nodes, providing a structure to represent entities & their relationships.

    – Tại sao lại là đồ thị? Câu hỏi đầu tiên chúng ta cần giải quyết: tại sao lại quan tâm đến đồ thị ngay từ đầu? Lý thuyết đồ thị, nghiên cứu toán học về đồ thị, đã nổi lên như 1 công cụ cơ bản để hiểu các hệ thống phức tạp & các mối quan hệ. Đồ thị là biểu diễn trực quan của 1 tập hợp các nút (còn gọi là đỉnh) & các cạnh kết nối các nút này, cung cấp 1 cấu trúc để biểu diễn các thực thể & các mối quan hệ của chúng.

    By representing a complex system as a network of entities with interactions, can analyze their relationships, allowing us to gain a deeper understanding of their underlying structures & patterns. Versatility of graphs makes them a popular choice in various domains, including:

    * CS, where graphs can be used to model structure of computer programs, making it easier to understand how different components of a system interact with each other
    * Physics, where graphs can be used to model physical systems & their interactions, e.g. relationship between particles & their properties
    * Biology, where graphs can be used to model biological systems, e.g. metabolic pathways, as a network of interconnected entities
    * Social sciences, where graphs can be used to study & understand complex social networks, including relationships between individuals in a community
    * Finance, where graphs can be used to analyze stock market trends & relationships between different financial instruments
    * Engineering, where graphs can be used to model & analyze complex systems, e.g. transportation networks & electrical power grids

    These domains naturally exhibit a relational structure. E.g., graphs are a natural representation of social networks: nodes are users, & edges represent friendships. But graphs are so versatile they can also be applied to domains where relationship structure is less natural, unlocking new insights & understanding.

    – Bằng cách biểu diễn 1 hệ thống phức tạp như 1 mạng lưới các thực thể có tương tác, chúng ta có thể phân tích các mối quan hệ của chúng, cho phép chúng ta hiểu sâu hơn về cấu trúc & các mô hình cơ bản của chúng. Tính linh hoạt của đồ thị khiến chúng trở thành lựa chọn phổ biến trong nhiều lĩnh vực, bao gồm:

    * Khoa học máy tính, trong đó đồ thị có thể được sử dụng để mô hình hóa cấu trúc của các chương trình máy tính, giúp dễ dàng hiểu được cách các thành phần khác nhau của 1 hệ thống tương tác với nhau.
    * Vật lý, trong đó đồ thị có thể được sử dụng để mô hình hóa các hệ thống vật lý & các tương tác của chúng, e.g.: mối quan hệ giữa các hạt & các tính chất của chúng
    * Sinh học, trong đó đồ thị có thể được sử dụng để mô hình hóa các hệ thống sinh học, e.g.: các con đường trao đổi chất, như 1 mạng lưới các thực thể được kết nối với nhau
    * Khoa học xã hội, trong đó đồ thị có thể được sử dụng để nghiên cứu & hiểu các mạng lưới xã hội phức tạp, bao gồm các mối quan hệ giữa các cá nhân trong 1 cộng đồng
    * Tài chính, trong đó đồ thị có thể được sử dụng để phân tích xu hướng thị trường chứng khoán & mối quan hệ giữa các công cụ tài chính khác nhau
    * Kỹ thuật, trong đó đồ thị có thể được sử dụng để mô hình hóa & phân tích các hệ thống phức tạp, e.g.: Mạng lưới giao thông & lưới điện

    Các miền này tự nhiên thể hiện 1 cấu trúc quan hệ. E.g., đồ thị là 1 biểu diễn tự nhiên của mạng xã hội: các nút là người dùng, & các cạnh biểu diễn tình bạn. Tuy nhiên, đồ thị rất linh hoạt nên chúng cũng có thể được áp dụng cho các miền mà cấu trúc quan hệ ít tự nhiên hơn, mở ra những hiểu biết mới & hiểu biết sâu sắc.

E.g., images can be represented as a graph, as in Fig. 1.2: Original image vs. graph representation of this image. Each pixel is a node, & edges represent relationships between neighboring pixels. This allows for application of graph-based algorithms to image processing & computer vision tasks.

– E.g., hình ảnh có thể được biểu diễn dưới dạng đồ thị, như trong Hình 1.2: Ảnh gốc so với biểu diễn đồ thị của ảnh này. Mỗi pixel là 1 nút, & các cạnh biểu diễn mối quan hệ giữa các pixel lân cận. Điều này cho phép áp dụng các thuật toán dựa trên đồ thị vào xử lý ảnh & các tác vụ thị giác máy tính.

Similarly, a sentence can be transformed into a graph, where nodes are words & edges represent relationships between adjacent words. This approach is useful in natural language processing & information retrieval tasks, where context & meaning of words are critical factors.

– Tại sao lại học đồ thị? Tương tự, 1 câu có thể được chuyển đổi thành đồ thị, trong đó các nút là các từ & các cạnh biểu diễn mối quan hệ giữa các từ liền kề. Cách tiếp cận này hữu ích trong xử lý ngôn ngữ tự nhiên & các tác vụ truy xuất thông tin, trong đó ngữ cảnh & ý nghĩa của từ là các yếu tố quan trọng.

Unlike text & images, graphs do not have a fixed structure. However, this flexibility also makes graphs more challenging to handle. Absence of a fixed structure means they can have an arbitrary number of nodes & edges, with no specific ordering. In addition, graphs can represent dynamic data, where connections between entities can change over time. E.g., relationships between users & products can change as they interact with each other. In this scenario, nodes & edges are updated to reflect changes in real world, e.g. new users, new products, & new relationships. In next sect, delve deeper into how to use graphs with ML to create valuable applications.

– Không giống như văn bản & hình ảnh, đồ thị không có cấu trúc cố định. Tuy nhiên, tính linh hoạt này cũng khiến việc xử lý đồ thị trở nên khó khăn hơn. Việc thiếu cấu trúc cố định đồng nghĩa với việc chúng có thể có số lượng nút & cạnh tùy ý, không theo thứ tự cụ thể. Ngoài ra, đồ thị có thể biểu diễn dữ liệu động, trong đó các kết nối giữa các thực thể có thể thay đổi theo thời gian. Ví dụ: mối quan hệ giữa người dùng & sản phẩm có thể thay đổi khi chúng tương tác với nhau. Trong trường hợp này, các nút & cạnh được cập nhật để phản ánh những thay đổi trong thế giới thực, e.g.: người dùng mới, sản phẩm mới, & mối quan hệ mới. Trong phần tiếp theo, tìm hiểu sâu hơn về cách sử dụng đồ thị với Học máy (ML) để tạo ra các ứng dụng có giá trị.

- 1.2. Why graph learning? Graph learning is application of ML techniques to graph data. This study area encompasses a range of tasks aimed at understanding & manipulating graph-structured data. There are many graphs learning tasks, including:
  - **Node classification** is a task that involves predicting category (class) of a node in a graph. E.g., it can categorize online users or items based on their characteristics. In this task, model is trained on a set of labeled nodes & their attributes, & it uses this information to predict class of unlabeled nodes.
  - **Link prediction** is a task that involves predicting missing links between pairs of nodes in a graph. This is useful in knowledge graph completion, where goal: complete a graph of entities & their relationships. E.g., it can be used to predict relationships between people based on their social network connections (friend recommendation).
  - **Graph classification** is a task that involves categorizing different graphs into predefined categories. 1 example of this is in molecular biology, where molecular structures can be represented as graphs, & goal: predict their properties for drug design. In this task, model is trained on a set of labeled graphs & their attributes, & it uses this information to categorize unseen graphs.
  - **Graph generation** is a task that involves generating new graphs based on a set of desired properties. 1 of main applications is generating novel molecular structures for drug discovery. This is achieved by training a model on a set of existing molecular structures & then using it to generate new, unseen structures. Generated structures can be evaluated for their potential as drug candidates & further studied.

– Học đồ thị là ứng dụng các kỹ thuật ML vào dữ liệu đồ thị. Lĩnh vực nghiên cứu này bao gồm 1 loạt các nhiệm vụ nhằm mục đích hiểu & thao tác dữ liệu có cấu trúc đồ thị. Có nhiều nhiệm vụ học đồ thị, bao gồm:

- **Phân loại nút** là 1 nhiệm vụ liên quan đến việc dự đoán loại (lớp) của 1 nút trong đồ thị. Ví dụ: nó có thể phân loại người dùng hoặc mục trực tuyến dựa trên các đặc điểm của họ. Trong nhiệm vụ này, mô hình được huấn luyện trên 1 tập hợp các nút được gắn nhãn & thuộc tính của chúng, & nó sử dụng thông tin này để dự đoán loại của các nút chưa được gắn nhãn.
- **Dự đoán liên kết** là 1 nhiệm vụ liên quan đến việc dự đoán các liên kết bị thiếu giữa các cặp nút trong đồ thị. Điều này hữu ích trong việc hoàn thành đồ thị tri thức, với mục tiêu: hoàn thành đồ thị các thực thể & mối quan hệ của chúng. Ví dụ: nó có thể được sử dụng để dự đoán mối quan hệ giữa mọi người dựa trên kết nối mạng xã hội của họ (khuyến nghị bạn bè).
- **Phân loại đồ thị** là 1 nhiệm vụ liên quan đến việc phân loại các đồ thị khác nhau thành các loại được xác định trước. 1 ví dụ về điều này là trong sinh học phân tử, nơi các cấu trúc phân tử có thể được biểu diễn dưới dạng đồ thị, & mục tiêu: dự đoán các đặc tính của chúng để thiết kế thuốc. Trong nhiệm vụ này, mô hình được huấn luyện trên 1 tập hợp các đồ thị được gắn nhãn & các thuộc tính của chúng, & nó sử dụng thông tin này để phân loại các đồ thị chưa được biết đến.
- **Tạo đồ thị** là 1 nhiệm vụ liên quan đến việc tạo ra các đồ thị mới dựa trên 1 tập hợp các đặc tính mong muốn. 1 trong những ứng dụng chính là tạo ra các cấu trúc phân tử mới để khám phá thuốc. Điều này đạt được bằng cách huấn luyện 1 mô hình trên 1 tập hợp các cấu trúc phân tử hiện có & sau đó sử dụng nó để tạo ra các cấu trúc mới, chưa được biết đến. Các cấu trúc được tạo ra có thể được đánh giá về tiềm năng của chúng như các ứng cử viên thuốc & nghiên cứu thêm.

Graph learning has many other practical applications that can have a significant impact. 1 of most well-known applications is *recommender systems*, where graph learning algorithms recommend relevant items to users based on their previous

interactions & relationships with other items. Another important application is *traffic forecasting*, where graph learning can improve travel time predictions by considering complex relationships between different routes & modes of transportation.

– Học đồ thị có nhiều ứng dụng thực tế khác có thể mang lại tác động đáng kể. 1 trong những ứng dụng nổi tiếng nhất là hệ thống đề xuất, trong đó các thuật toán học đồ thị đề xuất các mục liên quan cho người dùng dựa trên các tương tác trước đó của họ & mối quan hệ với các mục khác. 1 ứng dụng quan trọng khác là dự báo giao thông, trong đó học đồ thị có thể cải thiện dự đoán thời gian di chuyển bằng cách xem xét các mối quan hệ phức tạp giữa các tuyến đường & phương thức vận tải khác nhau.

Versatility & potential of graph learning make it an exciting field of research & development. Study of graphs have advanced rapidly in recent years, driven by availability of large datasets, powerful computing resources, & advancements in ML & AI. As a result, can list 4 prominent families of graph learning techniques [1]:

1. **Graph signal processing**, which applies traditional signal processing methods to graphs, e.g. graph Fourier transform & spectral analysis. These techniques reveal intrinsic properties of graph, e.g. its connectivity & structure.
2. **Matrix factorization**, which seeks to find low-dimensional representations of large matrices. Goal of matrix factorization: identify latent factors or patterns that explain observed relationships in original matrix. This approach can provide a compact & interpretable representation of data.
3. **Random walk**, which refers to a mathematical concept used to model movement of entities in a graph. By simulating random walks over a graph, information about relationships between nodes can be gathered. This is why they are often used to generate training data for ML models.
4. **DL**, which is a subfield of ML that focuses on neural networks with multiple layers. DL methods can effectively encode & represent graph data as vectors. These vectors can then be used in various tasks with remarkable performance.

– Tính linh hoạt & tiềm năng của học đồ thị khiến nó trở thành 1 lĩnh vực nghiên cứu & phát triển thú vị. Nghiên cứu về đồ thị đã phát triển nhanh chóng trong những năm gần đây, nhờ vào sự sẵn có của các tập dữ liệu lớn, tài nguyên điện toán mạnh mẽ, & những tiến bộ trong ML & AI. Do đó, có thể liệt kê 4 nhóm kỹ thuật học đồ thị nổi bật [1]:

1. **Xử lý tín hiệu đồ thị**, áp dụng các phương pháp xử lý tín hiệu truyền thống vào đồ thị, e.g.: biến đổi Fourier đồ thị & phân tích phổ. Các kỹ thuật này tiết lộ các đặc tính nội tại của đồ thị, e.g.: tính kết nối & cấu trúc của nó.
2. **Phân tích ma trận**, tìm cách tìm ra các biểu diễn chiều thấp của các ma trận lớn. Mục tiêu của phân tích ma trận: xác định các yếu tố hoặc mô hình tiềm ẩn giải thích các mối quan hệ quan sát được trong ma trận gốc. Phương pháp này có thể cung cấp 1 biểu diễn dữ liệu & có thể diễn giải được.
3. **Bước ngẫu nhiên**, đề cập đến 1 khái niệm toán học được sử dụng để mô hình hóa chuyển động của các thực thể trong đồ thị. Bằng cách mô phỏng các bước ngẫu nhiên trên đồ thị, thông tin về mối quan hệ giữa các nút có thể được thu thập. Đây là lý do tại sao chúng thường được sử dụng để tạo dữ liệu huấn luyện cho các mô hình ML.
4. **DL**, 1 lĩnh vực con của ML tập trung vào các mạng nơ-ron nhiều lớp. Các phương pháp DL có thể mã hóa hiệu quả & biểu diễn dữ liệu đồ thị dưới dạng vectơ. Các vectơ này sau đó có thể được sử dụng trong nhiều tác vụ khác nhau với hiệu suất đáng kể.

Important: these techniques are not mutually exclusive & often overlap in their applications. In practice, they are often combined to form hybrid models that leverage strengths of each. E.g., matrix factorization & DL techniques might be used in combination to learn low-dimensional representations of graph-structured data.

– Điều quan trọng cần lưu ý: các kỹ thuật này không loại trừ lẫn nhau & thường chồng chéo trong ứng dụng. Trong thực tế, chúng thường được kết hợp để tạo thành các mô hình lai tận dụng thế mạnh của từng kỹ thuật. Ví dụ: kỹ thuật phân tích ma trận & DL có thể được sử dụng kết hợp để học các biểu diễn dữ liệu có cấu trúc đồ thị ít chiều.

As delve into world of graph learning, crucial to understand fundamental building block of any ML technique: dataset. Traditional tabular datasets, e.g. spreadsheets, represent data as rows & columns with each row representing a single data point. However, in many real-world scenarios, relationships between data points are just as meaningful as data points themselves. This is where graph datasets come in. Graph datasets represent data points as nodes in a graph & relationships between those data points as edges.

Fig. 1.3: Family tree as a tabular dataset vs. a graph dataset: this dataset represents information about 5 members of a family. Each member has 3 features (or attributes): name, age, & gender. However, tabular version of this dataset does not show connections between these people. On contrary, graph version represents them with edges, which allows us to understand relationships in this family. In many contexts, connections between nodes are crucial in understanding data, which is why representing data in graph form is becoming increasingly popular.

– Hình 1.3: Cây phả hệ dưới dạng tập dữ liệu bảng so với tập dữ liệu đồ thị: tập dữ liệu này biểu diễn thông tin về 5 thành viên trong 1 gia đình. Mỗi thành viên có 3 đặc điểm (hoặc thuộc tính): tên, tuổi, & giới tính. Tuy nhiên, phiên bản bảng của tập dữ liệu này không thể hiện mối liên hệ giữa những người này. Ngược lại, phiên bản đồ thị biểu diễn họ bằng các cạnh, cho phép chúng ta hiểu các mối quan hệ trong gia đình này. Trong nhiều bối cảnh, mối liên hệ giữa các nút rất quan trọng để hiểu dữ liệu, đó là lý do tại sao việc biểu diễn dữ liệu dưới dạng đồ thị ngày càng trở nên phổ biến.

Now have a basic understanding of graph ML & different types of tasks it involves, can move on to exploring 1 of most important approaches for solving these tasks: GNNs.

- 1.3. Why GNNs? In this book, focus on DL family of graph learning techniques, often referred to as graph neural networks. GNNs are a new category of DL architecture & are specifically designed for graph-structured data. Unlike traditional DL algorithms, which have been primarily developed for text & images, GNNs are explicitly made to process & analyze graph

dataset Fig. 1.4: High-level architecture of a GNN pipeline, with a graph as input & an output that corresponds to a given task: a. Node classification. b. Link prediction. c. Graph classification.

– Trong cuốn sách này, chúng tôi tập trung vào họ kỹ thuật học đồ thị DL, thường được gọi là mạng nơ-ron đồ thị. GNN là 1 loại kiến trúc DL mới & được thiết kế riêng cho dữ liệu có cấu trúc đồ thị. Không giống như các thuật toán DL truyền thống, vốn chủ yếu được phát triển cho văn bản & hình ảnh, GNN được thiết kế rõ ràng để xử lý & phân tích tập dữ liệu đồ thị Hình 1.4: Kiến trúc cấp cao của 1 đường ống GNN, với đồ thị làm đầu vào & đầu ra tương ứng với 1 tác vụ nhất định: a. Phân loại nút. b. Dự đoán liên kết. c. Phân loại đồ thị.

GNNs have emerged as a powerful tool for graph learning & have shown excellent results in various tasks & industries. 1 of most striking examples is how a CNN model identified a new antibiotic [2]. Model was trained on 2500 molecules & was tested on a library of 6000 compounds. It predicted that a molecule called halicin should be able to kill many antibiotic-resistant bacteria while having low toxicity to human cells. Based on this prediction, researchers used halicin to treat mice infected with antibiotic-resistant bacteria. They demonstrated its effectiveness & believe model could be used to design new drugs.

– Mạng nơ-ron nhân tạo (GNN) đã nổi lên như 1 công cụ mạnh mẽ cho việc học đồ thị & đã cho thấy kết quả xuất sắc trong nhiều nhiệm vụ & ngành công nghiệp. 1 trong những ví dụ nổi bật nhất là cách 1 mô hình CNN xác định 1 loại kháng sinh mới [2]. Mô hình được huấn luyện trên 2500 phân tử & đã được thử nghiệm trên thư viện gồm 6000 hợp chất. Nó dự đoán rằng 1 phân tử có tên là halicin có thể tiêu diệt nhiều loại vi khuẩn kháng kháng sinh trong khi có độc tính thấp đối với tế bào người. Dựa trên dự đoán này, các nhà nghiên cứu đã sử dụng halicin để điều trị cho chuột bị nhiễm vi khuẩn kháng kháng sinh. Họ đã chứng minh hiệu quả của nó & tin rằng mô hình có thể được sử dụng để thiết kế các loại thuốc mới.

How do GNNs work? Take example of a node classification task in a social network, like previous family tree. In a node classification task, GNNs take advantage of information from different sources to create a vector representation of each node in graph. This representation encompasses not only original node features (e.g. name, age, & gender) but also information from edge features (e.g. strength of relationships between nodes) & global features (e.g. network-wide statistics).

– Mạng GNN hoạt động như thế nào? Hãy lấy ví dụ về tác vụ phân loại nút trong mạng xã hội, chẳng hạn như cây phả hệ trước đây. Trong tác vụ phân loại nút, GNN tận dụng thông tin từ các nguồn khác nhau để tạo biểu diễn vectơ cho mỗi nút trong đồ thị. Biểu diễn này không chỉ bao gồm các đặc điểm nút gốc (e.g.: tên, tuổi, & giới tính) mà còn bao gồm thông tin từ các đặc điểm biên (e.g.: cường độ mối quan hệ giữa các nút) & các đặc điểm toàn cục (e.g.: thống kê toàn mạng).

This is why GNNs are more efficient than traditional ML techniques on graphs. Instead of being limited to original attributes, GNNs enrich original node features with attributes from neighboring nodes, edges, & global features, making representation much more comprehensive & meaningful. New node representations are then used to perform a specific task, e.g. node classification, regression, or link prediction.

– Đây là lý do tại sao GNN hiệu quả hơn các kỹ thuật ML truyền thống trên đồ thị. Thay vì bị giới hạn ở các thuộc tính gốc, GNN làm giàu các đặc trưng nút gốc bằng các thuộc tính từ các nút lân cận, cạnh, & đặc trưng toàn cục, giúp biểu diễn trở nên toàn diện hơn & có ý nghĩa hơn nhiều. Các biểu diễn nút mới sau đó được sử dụng để thực hiện 1 tác vụ cụ thể, e.g.: phân loại nút, hồi quy hoặc dự đoán liên kết.

Specifically, GNNs define a graph convolution operation that aggregates information from neighboring nodes & edges to update node representation. This operation is performed iteratively, allowing model to learn more complex relationships between nodes as number of iterations increases. E.g., Fig. 1.5: Input graph vs. computation graph representing how a GNN computes representation of node 5 based on its neighbors. shows how a GNN would calculate representation of node 5 using neighboring nodes.

– Cụ thể, GNN định nghĩa 1 phép toán tích chập đồ thị tổng hợp thông tin từ các nút & cạnh lân cận để cập nhật biểu diễn nút. Phép toán này được thực hiện lặp đi lặp lại, cho phép mô hình học các mối quan hệ phức tạp hơn giữa các nút khi số lần lặp tăng lên. Ví dụ: Hình 1.5: Đồ thị đầu vào so với đồ thị tính toán biểu diễn cách GNN tính toán biểu diễn của nút 5 dựa trên các nút lân cận. cho thấy cách GNN tính toán biểu diễn của nút 5 bằng cách sử dụng các nút lân cận.

Worth noting: Fig. 1.5 provides a simplified illustration of a computation graph. In reality, there are various kinds of GNNs & GNN layers, each of which has a unique structure & way of aggregating information from neighboring nodes. These different variants of GNNs also have their own advantages & limitations & are well-suited for specific types of graph data & tasks. When selecting appropriate GNN architecture for a particular problem, crucial to understand characteristics of graph data & desired outcome.

– Lưu ý: Hình 1.5 cung cấp 1 minh họa đơn giản về đồ thị tính toán. Trên thực tế, có nhiều loại GNN & các lớp GNN, mỗi loại có 1 cấu trúc riêng & cách tổng hợp thông tin từ các nút lân cận. Các biến thể GNN khác nhau này cũng có những ưu điểm & hạn chế riêng & phù hợp với các loại dữ liệu đồ thị & tác vụ cụ thể. Khi lựa chọn kiến trúc GNN phù hợp cho 1 bài toán cụ thể, điều quan trọng là phải hiểu các đặc điểm của dữ liệu đồ thị & kết quả mong muốn.

More generally, GNNs, like other DL techniques, are most effective when applied to specific problems. These problems are characterized by high complexity, i.e., learning good representations is critical to solving task at hand. E.g., a highly complex task could be recommending right products among billions of options to millions of customers. On other hand, some problems, e.g. finding youngest member of our family tree, can be solved without any ML technique.

– Nói chung, GNN, giống như các kỹ thuật DL khác, hiệu quả nhất khi được áp dụng cho các vấn đề cụ thể. Những vấn đề này được đặc trưng bởi độ phức tạp cao, i.e., việc học các biểu diễn tốt là rất quan trọng để giải quyết nhiệm vụ. E.g., 1 nhiệm vụ cực kỳ phức tạp có thể là đề xuất sản phẩm phù hợp giữa hàng tỷ lựa chọn cho hàng triệu khách hàng. Mặt khác, 1 số vấn đề, e.g. tìm thành viên trẻ nhất trong cây phả hệ gia đình, có thể được giải quyết mà không cần bất kỳ kỹ thuật ML nào.

Furthermore, GNNs require a substantial amount of data to perform effectively. Traditional ML techniques might be a better fit in cases where dataset is small, as they are less reliant on large amounts of data. However, these techniques do not scale as well as GNNs. GNNs can process bigger datasets thanks to parallel & distributed training. They can also exploit additional information more efficiently, which produces better results.

– Hơn nữa, GNN cần 1 lượng dữ liệu đáng kể để hoạt động hiệu quả. Các kỹ thuật ML truyền thống có thể phù hợp hơn trong trường hợp tập dữ liệu nhỏ, vì chúng ít phụ thuộc vào lượng dữ liệu lớn. Tuy nhiên, các kỹ thuật này không có khả năng mở rộng tốt như GNN. GNN có thể xử lý các tập dữ liệu lớn hơn nhờ đào tạo song song & phân tán. Chúng cũng có thể khai thác thông tin bổ sung hiệu quả hơn, mang lại kết quả tốt hơn.

- Summary. In this chap, answered 3 main questions: why graphs, why graph learning, & why GNNs? 1st, explored versatility of graphs in representing various data types, e.g. social networks & transportation networks, but also text & images. Discussed different applications of graph learning, including node classification & graph classification, & highlighted 4 main families of graph learning techniques. Finally, emphasized significance of GNNs & their superiority over other techniques, especially regarding large, complex datasets. By answering these 3 main questions, aimed to provide a comprehensive overview of importance of GNNs & why they are becoming vital tools in ML.

  – Trong chương này, chúng tôi đã trả lời 3 câu hỏi chính: tại sao lại dùng đồ thị, tại sao lại dùng học đồ thị, & tại sao lại dùng mạng lưới thần kinh nhân tạo (GNN)? Đầu tiên, chúng tôi khám phá tính linh hoạt của đồ thị trong việc biểu diễn các loại dữ liệu khác nhau, e.g. mạng xã hội & mạng lưới giao thông, cũng như văn bản & hình ảnh. Chúng tôi đã thảo luận về các ứng dụng khác nhau của học đồ thị, bao gồm phân loại nút & phân loại đồ thị, & làm nổi bật 4 nhóm kỹ thuật học đồ thị chính. Cuối cùng, chúng tôi nhấn mạnh tầm quan trọng của mạng lưới thần kinh nhân tạo (GNN) & sự vượt trội của chúng so với các kỹ thuật khác, đặc biệt là đối với các tập dữ liệu lớn & phức tạp. Bằng cách trả lời 3 câu hỏi chính này, chúng tôi mong muốn cung cấp 1 cái nhìn tổng quan toàn diện về tầm quan trọng của mạng lưới thần kinh nhân tạo (GNN) & lý do tại sao chúng đang trở thành những công cụ thiết yếu trong học máy.

  In Chap. 2: Graph Theory for GNNs, dive deeper into basics of graph theory, which provides foundation for understanding GNNs. This chap will cover fundamental concepts of graph theory, including concepts e.g. adjacency matrices & degrees. Additionally, delve into different types of graphs & their applications, e.g. directed & undirected graphs, & weighted & unweighted graphs.

  – Trong Chương 2: Lý thuyết Đồ thị cho Mạng Lưới Hàm Truyền Thống (GNN), tìm hiểu sâu hơn về những kiến thức cơ bản của lý thuyết đồ thị, vốn là nền tảng để hiểu về GNN. Chương này sẽ đề cập đến các khái niệm cơ bản của lý thuyết đồ thị, bao gồm các khái niệm như ma trận kề & bậc. Ngoài ra, tìm hiểu sâu hơn về các loại đồ thị khác nhau & ứng dụng của chúng, e.g. đồ thị có hướng & vô hướng, & trọng số & không trọng số.

- 2. Graph Theory for Graph Neural Networks. Graph theory is a fundamental branch of mathematics that deals with study of graphs & networks. A graph is a visual representation of complex data structures that helps us understand relationships between different entities. Graph theory provides us with tools to model & analyze a vast array of real-world problems, e.g. transportation systems, social networks, & internet connectivity.

  – Lý thuyết đồ thị là 1 nhánh cơ bản của toán học, chuyên nghiên cứu về đồ thị & mạng. Đồ thị là 1 biểu diễn trực quan của các cấu trúc dữ liệu phức tạp, giúp chúng ta hiểu được mối quan hệ giữa các thực thể khác nhau. Lý thuyết đồ thị cung cấp cho chúng ta các công cụ để mô hình hóa & phân tích 1 loạt các vấn đề thực tế, e.g. hệ thống giao thông, mạng xã hội, & kết nối internet.

  In this chap, delve into essentials of graph theory, covering 3 main topics: graph properties, graph concepts, & graph algorithms. Begin by defining graphs & their components. Then introduce different types of graphs & explain their properties & applications. Next, cover fundamental graph concepts, objects, & measures, including adjacency matrix. Finally, dive into graph algorithms, focusing on 2 fundamental algorithms, BFS & DFS. By end of this chap, have a solid foundation in graph theory, allowing you to tackle more advanced topics & design GNNs. In this chap, cover 3 main topics: introducing graph properties, discovering graph concepts, exploring graph algorithms.

  – Trong chương này, chúng ta sẽ đi sâu vào những kiến thức cốt lõi của lý thuyết đồ thị, bao gồm 3 chủ đề chính: tính chất đồ thị, khái niệm đồ thị & thuật toán đồ thị. Bắt đầu bằng việc định nghĩa đồ thị & các thành phần của chúng. Sau đó, giới thiệu các loại đồ thị khác nhau & giải thích tính chất cũng như ứng dụng của chúng. Tiếp theo, chúng ta sẽ tìm hiểu các khái niệm cơ bản về đồ thị, đối tượng, độ đo, bao gồm cả ma trận kề. Cuối cùng, chúng ta sẽ đi sâu vào các thuật toán đồ thị, tập trung vào 2 thuật toán cơ bản: BFS & DFS. Kết thúc chương này, bạn sẽ có nền tảng vững chắc về lý thuyết đồ thị, cho phép bạn giải quyết các chủ đề nâng cao hơn & thiết kế mạng nơ-ron nhân tạo (GNN). Trong chương này, chúng ta sẽ tìm hiểu 3 chủ đề chính: giới thiệu tính chất đồ thị, khám phá các khái niệm đồ thị & tìm hiểu thuật toán đồ thị.

  - 2.1. Introducing graph properties. In graph theory, a graph is a mathematical structure consisting of a set of objects, called *vertices* or *nodes*, & a set of connections, called *edges*, which link pairs of vertices. Notation $G = (V, E)$ is used to represent a graph, where G is graph, V: set of vertices, E: set of edges. Nodes of a graph can represent any objects, e.g. cities, people, web pages, or molecules, & edges represent relationships or connections between them, e.g. physical roads, social relationships, hyperlinks, or chemical bonds. This sect provides an overview of fundamental graph properties used extensively in later chaps.

    – Giới thiệu các tính chất của đồ thị. Trong lý thuyết đồ thị, đồ thị là 1 cấu trúc toán học bao gồm 1 tập hợp các đối tượng, được gọi là *đỉnh* hoặc *nút*, & 1 tập hợp các kết nối, được gọi là *cạnh*, nối các cặp đỉnh. Ký hiệu $G = (V, E)$ được sử dụng để biểu diễn 1 đồ thị, trong đó G là đồ thị, V: tập hợp các đỉnh, E: tập hợp các cạnh. Các nút của đồ thị có thể biểu diễn bất

kỳ đối tượng nào, e.g.: thành phố, con người, trang web hoặc phân tử, & các cạnh biểu diễn các mối quan hệ hoặc kết nối giữa chúng, e.g.: đường xá thực tế, mối quan hệ xã hội, siêu liên kết hoặc liên kết hóa học. Phần này cung cấp tổng quan về các tính chất cơ bản của đồ thị được sử dụng rộng rãi trong các chương sau.

- Directed graphs. 1 of most basic properties of a graph is whether it is directed or undirected. In a *directed graph*, also called a *digraph*, each edge has a direction or orientation, i.e., edge connects 2 nodes in a particular direction, where 1 node is source & other is destination. In contrast, an undirected graph has undirected edges, where edges have no direction, i.e., edge between 2 vertices can be traversed in either direction, & order in which we visit nodes does not matter. In Python, can use `networkx` library to define an undirected graph as follows with `nx.Graph()`:

  – 1 trong những đặc điểm cơ bản nhất của đồ thị là nó có hướng hay vô hướng. Trong 1 đồ thị có hướng, còn được gọi là đồ thị có hướng, mỗi cạnh có 1 hướng hoặc hướng, i.e., cạnh nối 2 nút theo 1 hướng cụ thể, trong đó 1 nút là nguồn & nút còn lại là đích. Ngược lại, 1 đồ thị vô hướng có các cạnh vô hướng, trong đó các cạnh không có hướng, i.e., cạnh giữa 2 đỉnh có thể được duyệt theo cả 2 hướng, & thứ tự chúng ta duyệt các nút không quan trọng. Trong Python, có thể sử dụng thư viện `networkx` để định nghĩa 1 đồ thị vô hướng như sau với `nx.Graph()`:

```
import networkx as nx
# define an undirected graph
G = nx.Graph()
G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
```

  Code to create a directed graph is similar; simply replace `nx.Graph()` with `nx.DiGraph()`:

```
# define a directed graph
DG = nx.DiGraph()
DG.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
```

  In directed graphs, edges are typically represented using arrows to denote their orientation.

- Weighted graphs. Another important property of graphs is whether edges are weighted or unweighted. In a *weighted graph*, each edge has a weight or cost associated with it. These weights can represent various factors, e.g. distance, travel time, or cost.

  – 1 thuộc tính quan trọng khác của đồ thị là các cạnh có trọng số hay không có trọng số. Trong 1 đồ thị có trọng số, mỗi cạnh đều có trọng số hoặc chi phí liên quan. Các trọng số này có thể biểu thị nhiều yếu tố khác nhau, e.g.: khoảng cách, thời gian di chuyển hoặc chi phí.

  E.g., in a transportation network, weights of edges might represent distances between different cities or time it takes to travel between them. In contrast, unweighted graphs have no weight associated with their edges. These types of graphs are commonly used in situations where relationships between nodes are binary, & edges simply indicate presence or absence of a connection between them.

  – E.g., trong mạng lưới giao thông, trọng số của các cạnh có thể biểu thị khoảng cách giữa các thành phố khác nhau hoặc thời gian di chuyển giữa chúng. Ngược lại, đồ thị không trọng số không gắn trọng số với các cạnh của chúng. Các loại đồ thị này thường được sử dụng trong các trường hợp mối quan hệ giữa các nút là nhị phân, & cạnh chỉ đơn giản biểu thị sự có hoặc không có kết nối giữa chúng.

  Can modify previous undirected graph to add weights to our edges. In `networkx`, edges of graph are defined with a tuple containing start & end nodes & a dictionary specifying edge's weight:

```
# define an weighted graph
WG = nx.Graph()
WG.add_edges_from([('A', 'B', {"weight": 10}), ('A', 'C', {"weight": 20}), ('B', 'D', {"weight": 30}), ('B',
labels = nx.get_edge_attributes(WG, "weight")
```

- Connected graphs. Graph connectivity is a fundamental concept in graph theory that is closely related to graph's structure & function.

  – Kết nối đồ thị là 1 khái niệm cơ bản trong lý thuyết đồ thị có liên quan chặt chẽ đến cấu trúc & chức năng của đồ thị.

  In a connected graph, there is a path between any 2 vertices in graph. Formally, a graph G is connected iff for every pair of vertices u, v in G, there exists a path from u to v. In contrast, a graph is disconnected if it is not connected, i.e., at least 2 vertices are not connected by a path.

  – Trong 1 đồ thị liên thông, luôn có 1 đường đi giữa 2 đỉnh bất kỳ trong đồ thị. Về mặt hình thức, 1 đồ thị G được coi là liên thông nếu & chỉ nếu (iff) với mọi cặp đỉnh u, v trong G, tồn tại 1 đường đi từ u đến v. Ngược lại, 1 đồ thị được coi là không liên thông nếu nó không liên thông, i.e., có ít nhất 2 đỉnh không được nối với nhau bằng 1 đường đi.

  `networkx` library provides a built-in function for verifying whether a graph is connected or not. In following example, 1st graph contains isolated nodes (4 & 5), unlike 2nd graph.

  – Thư viện `networkx` cung cấp 1 hàm tích hợp để kiểm tra xem đồ thị có kết nối hay không. Trong ví dụ sau, đồ thị thứ nhất chứa các nút riêng biệt (4 & 5), không giống như đồ thị thứ 2.

```
# verify whether a graph is connected or disconnected
G1 = nx.Graph()
G1.add_edges_from([(1, 2), (2, 3), (3, 1), (4, 5)])
print(f"Is graph 1 connected? {nx.is_connected(G1)}")

G2 = nx.Graph()
G2.add_edges_from([(1, 2), (2, 3), (3, 1), (1, 4)])
print(f"Is graph 2 connected? {nx.is_connected(G2)}")
```

This property is easy to visualize with small graphs. Connected graphs have several interesting properties & applications. E.g., in a communication network, a connected graph ensures that any 2 nodes can communicate with each other through a path. In contrast, disconnected graphs can have isolated nodes that cannot communicate with other nodes in network, making it challenging to design efficient routing algorithms.

– Tính chất này dễ hình dung bằng các đồ thị nhỏ. Đồ thị liên thông có 1 số tính chất & ứng dụng thú vị. E.g., trong mạng truyền thông, 1 đồ thị liên thông đảm bảo rằng bất kỳ 2 nút nào cũng có thể giao tiếp với nhau thông qua 1 đường dẫn. Ngược lại, đồ thị liên thông có thể có các nút bị cô lập không thể giao tiếp với các nút khác trong mạng, khiến việc thiết kế các thuật toán định tuyến hiệu quả trở nên khó khăn.

There are different ways to measure connectivity of a graph. 1 of most common measures is minimum number of edges that need to be removed to disconnect graph, which is known as graph's minimum cut. Minimum cut problem has several applications in network flow optimization, clustering, & community detection.

– Có nhiều cách khác nhau để đo lường khả năng kết nối của 1 đồ thị. 1 trong những phép đo phổ biến nhất là số cạnh tối thiểu cần loại bỏ để ngắt kết nối đồ thị, được gọi là đường cắt tối thiểu của đồ thị. Bài toán đường cắt tối thiểu có nhiều ứng dụng trong tối ưu hóa luồng mạng, phân cụm, & phát hiện cộng đồng.

- Types of graphs. In addition to commonly used graph types, there are some special types of graphs that have unique properties & characteristics:

1. A tree is a connected, undirected graph with no cycles. Since there is only 1 path between any 2 nodes in a tree, a tree is a special case of a graph. Trees are often used to model hierarchical structures, e.g. family trees, organizational structures, or classification trees.
2. A rooted tree is a tree in which 1 node is designated as root, & all other vertices are connected to it by a unique path. Rooted trees are often used in CS to represent hierarchical data structures, e.g. filesystems or structures of XML documents.
3. A directed acyclic graph (DAG) is a directed graph that has no cycles, i.e., edges can only be traversed in a particular direction, & there are no loops or cycles. DAGs are often used to model dependencies between tasks or events – e.g., in project management or in computing critical path of a job.
4. A bipartite graph is a graph in which vertices can be divided into 2 disjoint sets, s.t. all edges connect vertices in different sets. Bipartite graphs are often used in mathematics & CS to model relationships between 2 different types of objects, e.g. buyers & sellers, or employees & projects.
5. A complete graph is a graph in which every pair of vertices is connected by an edge. Complete graphs are often used in combinatorics to model problems involving all possible pairwise connections, & in computer networks to model fully connected networks.

Now have reviewed essential types of graphs, explore some of most important graph objects. Understanding these concepts will help analyze & manipulate graphs effectively.

– Các loại đồ thị. Ngoài các loại đồ thị thường dùng, còn có 1 số loại đồ thị đặc biệt với các thuộc tính & đặc điểm riêng:

1. Cây là đồ thị liên thông, vô hướng & không có chu trình. Vì chỉ có 1 đường đi giữa 2 nút bất kỳ trong cây, nên cây là 1 trường hợp đặc biệt của đồ thị. Cây thường được sử dụng để mô hình hóa các cấu trúc phân cấp, e.g.: cây phả hệ, cấu trúc tổ chức hoặc cây phân loại.
2. Cây có gốc là cây trong đó 1 nút được chỉ định là gốc, & tất cả các đỉnh khác được kết nối với nó bằng 1 đường đi duy nhất. Cây có gốc thường được sử dụng trong CS để biểu diễn các cấu trúc dữ liệu phân cấp, e.g.: hệ thống tệp hoặc cấu trúc của tài liệu XML.
3. Đồ thị có hướng phi chu trình (DAG) là đồ thị có hướng không có chu trình, i.e., các cạnh chỉ có thể được duyệt theo 1 hướng cụ thể, & không có vòng lặp hoặc chu trình. DAG thường được sử dụng để mô hình hóa sự phụ thuộc giữa các tác vụ hoặc sự kiện – e.g.: trong quản lý dự án hoặc tính toán đường dẫn tới hạn của 1 công việc.
4. Đồ thị 2 phía là đồ thị trong đó các đỉnh có thể được chia thành 2 tập rời rạc, nghĩa là tất cả các cạnh đều nối các đỉnh thuộc các tập khác nhau. Đồ thị 2 phía thường được sử dụng trong toán học & Khoa học Máy tính để mô hình hóa mối quan hệ giữa 2 loại đối tượng khác nhau, e.g.: người mua & người bán, hoặc nhân viên & dự án.
5. Đồ thị đầy đủ là đồ thị trong đó mỗi cặp đỉnh được nối bởi 1 cạnh. Đồ thị đầy đủ thường được sử dụng trong tổ hợp để mô hình hóa các bài toán liên quan đến tất cả các kết nối từng cặp có thể có, & trong mạng máy tính để mô hình hóa các mạng được kết nối đầy đủ.

Bây giờ chúng ta đã xem xét các loại đồ thị thiết yếu, khám phá 1 số đối tượng đồ thị quan trọng nhất. Việc hiểu các khái niệm này sẽ giúp phân tích & thao tác đồ thị 1 cách hiệu quả.

∘ Discovering graph concepts. In this sect, explore some of essential concepts in graph theory, including graph objects (e.g. degree & neighbors), graph measures (e.g., centrality & density), & adjacency matrix representation.

– Khám phá các khái niệm về đồ thị. Trong phần này, khám phá 1 số khái niệm thiết yếu trong lý thuyết đồ thị, bao gồm các đối tượng đồ thị (e.g.: bậc & lân cận), các phép đo đồ thị (e.g.: độ trung tâm & mật độ), & biểu diễn ma trận kề.

∗ Fundamental objects. 1 of key concepts in graph theory is degree of a node, which is number of edges incident to this node. An edge is said to be incident on a node if that node is 1 of edge's endpoints. Degree of a node v is often denoted by $\deg v$. It can be defined for both directed & undirected graphs:

· In an undirected graph, degree of a vertex is number of edges connected to it. If node is connected to itself (called a loop, or self-loop), it adds 2 to degree.

· In a directed graph, degree is divided into 2 types: indegree & outdegree. Indegree (denoted by $\deg\_(v)$) of a node represents number of edges that point towards that node, while outdegree (denoted by $\deg^+(v)$) represents number of edges that start from that node. In this case, a self-loop adds 1 to indegree & to outdegree.

Indegree & outdegree are essential for analyzing & understanding directed graphs, as they provide insight into how information or resources are distributed within graph. E.g., nodes with high indegree are likely to be important sources of information or resources. In contrast, nodes with high outdegree are likely to be important destinations or consumers of information or resources.

– Các đối tượng cơ bản. 1 trong những khái niệm chính trong lý thuyết đồ thị là bậc của 1 nút, i.e., số cạnh liên thuộc nút này. 1 cạnh được gọi là liên thuộc nút nếu nút đó là 1 trong các điểm cuối của cạnh đó. Bậc của 1 nút v thường được ký hiệu là $\deg v$. Nó có thể được định nghĩa cho cả đồ thị có hướng & vô hướng:

· Trong đồ thị vô hướng, bậc của 1 đỉnh là số cạnh được nối với nó. Nếu nút được nối với chính nó (gọi là vòng lặp, hoặc tự vòng lặp), nó cộng thêm 2 vào bậc.

· Trong đồ thị có hướng, bậc được chia thành 2 loại: bậc vào & bậc ra. Bậc vào (ký hiệu là $\deg\_(v)$) của 1 nút biểu diễn số cạnh hướng đến nút đó, trong khi bậc ra (ký hiệu là $\deg^+(v)$) biểu diễn số cạnh bắt đầu từ nút đó. Trong trường hợp này, 1 vòng lặp tự thêm 1 vào bậc vào & vào bậc ra.

Bậc vào & bậc ra rất cần thiết để phân tích & hiểu đồ thị có hướng, vì chúng cung cấp cái nhìn sâu sắc về cách thông tin hoặc tài nguyên được phân bổ trong đồ thị. Ví dụ: các nút có bậc vào cao có thể là nguồn thông tin hoặc tài nguyên quan trọng. Ngược lại, các nút có bậc ra cao có thể là đích đến hoặc người tiêu thụ thông tin hoặc tài nguyên quan trọng.

In `networkx`, can simply calculate node degree, indegree, or outdegree using built-in methods:

– Trong `networkx`, có thể dễ dàng tính toán bậc nút, bậc vào hoặc bậc ra bằng các phương pháp tích hợp:

```
# calculate node degree, indegree, or outdegree
G = nx.Graph()
G.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
print(f"deg(A) = {G.degree['A']}")

DG = nx.DiGraph()
DG.add_edges_from([('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E'), ('C', 'F'), ('C', 'G')])
print(f"deg^-(A) = {DG.in_degree['A']}")
print(f"deg^+(A) = {DG.out_degree['A']}")
```

Concept of node degree is related to that of neighbors. Neighbors refer to nodes directly connected to a particular node through an edge. Moreover, 2 nodes are said to be adjacent if they share at least 1 common neighbor. Concepts of neighbors & adjacency are fundamental to many graph algorithms & applications, e.g. searching for a path between 2 nodes or identifying clusters in a network.

– Khái niệm bậc nút liên quan đến khái niệm lân cận. Lân cận là các nút được kết nối trực tiếp với 1 nút cụ thể thông qua 1 cạnh. Hơn nữa, 2 nút được gọi là lân cận nếu chúng có chung ít nhất 1 lân cận. Khái niệm lân cận & kề là nền tảng cho nhiều thuật toán đồ thị & ứng dụng, e.g.: tìm kiếm đường đi giữa 2 nút hoặc xác định các cụm trong mạng.

In graph theory, a path is a sequence of edges that connects 2 nodes (or more) in a graph. Length of a path is number of edges that are traversed along path. There are different types of paths, but 2 of them are particularly important:

1. A simple path is a path that does not visit any node more than once, except for start & end vertices.
2. A cycle is a path in which 1st & last vertices are same. A graph is said to be acylic if it contains no cycles, e.g. trees & DAGs.

Degrees & paths can be used to determine importance of a node in a network. This measure is referred to as *centrality*.

– Trong lý thuyết đồ thị, đường đi là 1 chuỗi các cạnh nối 2 nút (hoặc nhiều hơn) trong 1 đồ thị. Độ dài của đường đi là số cạnh được đi qua dọc theo đường đi. Có nhiều loại đường đi khác nhau, nhưng có 2 loại đặc biệt quan trọng:

1. Đường đi đơn là đường đi không đi qua bất kỳ nút nào quá 1 lần, ngoại trừ đỉnh đầu & đỉnh cuối.
2. Chu trình là đường đi mà đỉnh đầu & đỉnh cuối giống nhau. 1 đồ thị được gọi là chu trình nếu nó không chứa chu trình nào, e.g.: cây & DAG.

Bậc & đường đi có thể được sử dụng để xác định tầm quan trọng của 1 nút trong mạng. Thước đo này được gọi là độ trung tâm.

∗ Graph measures. Centrality quantifies importance of a vertex or node in a network. It helps us to identify key node in a graph based on their connectivity & influence on flow of information or interactions within network. There are several measures of centrality, each providing a different perspective on importance of a node:

1. **Degree centrality** is 1 of simplest & most commonly used measures of centrality. It is simply defined as degree of node. A high degree centrality indicates a vertex is highly connected to other vertices in graph, & thus significantly influences network.
2. **Closeness centrality** measures how close a node is to all other nodes in graph. It corresponds to average length of shortest path between target node & all other nodes in graph. A node with high closeness centrality can quickly reach all other vertices in network.
3. **Betweenness centrality** measures number of times a node lies on shortest path between pairs of other nodes in graph. A node with high betweenness centrality acts as a bottleneck or bridge between different parts of graph.

Calculate these measure on previous graphs using built-in functions of `networkx` & analyze result:
– Đo lường đồ thị. Độ tập trung định lượng tầm quan trọng của 1 đỉnh hoặc nút trong mạng. Nó giúp chúng ta xác định nút chính trong đồ thị dựa trên khả năng kết nối & ảnh hưởng của chúng đến luồng thông tin hoặc tương tác trong mạng. Có 1 số thước đo độ tập trung, mỗi thước đo cung cấp 1 góc nhìn khác nhau về tầm quan trọng của 1 nút:

1. **Độ tập trung bậc** là 1 trong những thước đo độ tập trung đơn giản nhất & được sử dụng phổ biến nhất. Nó được định nghĩa đơn giản là bậc của nút. Độ tập trung bậc cao cho thấy 1 đỉnh có kết nối cao với các đỉnh khác trong đồ thị, & do đó ảnh hưởng đáng kể đến mạng.
2. **Độ tập trung gần** đo lường mức độ gần của 1 nút với tất cả các nút khác trong đồ thị. Nó tương ứng với độ dài trung bình của đường đi ngắn nhất giữa nút mục tiêu & tất cả các nút khác trong đồ thị. 1 nút có độ tập trung gần cao có thể nhanh chóng tiếp cận tất cả các đỉnh khác trong mạng.
3. **Độ tập trung giữa** đo lường số lần 1 nút nằm trên đường đi ngắn nhất giữa các cặp nút khác trong đồ thị. 1 nút có tính trung tâm cao đóng vai trò như 1 nút thắt cổ chai hoặc cầu nối giữa các phần khác nhau của đồ thị.

Tính toán các phép đo này trên các đồ thị trước đó bằng cách sử dụng các hàm tích hợp của `networkx` & phân tích kết quả:

```
# Graph measures
print(f"Degree centrality = {nx.degree_centrality(G)}")
print(f"Closeness centrality = {nx.closeness_centrality(G)}")
print(f"Betweenness centrality = {nx.betweenness_centrality(G)}")
```

Importance of nodes A, B, C in a graph depends on type of centrality used. Degree centrality considers nodes B & C to be more important because they have more neighbors than node A. However, in closeness centrality, node A is most important as it can reach any other node in graph in shortest possible path. On other hand, nodes A, B, & C have equal betweenness centrality, as they all lie on a large number of shortest paths between other nodes.
– Tầm quan trọng của các nút A, B, C trong đồ thị phụ thuộc vào loại trung tâm được sử dụng. Trung tâm bậc coi các nút B & C quan trọng hơn vì chúng có nhiều hàng xóm hơn nút A. Tuy nhiên, về trung tâm gần, nút A quan trọng nhất vì nó có thể đến bất kỳ nút nào khác trong đồ thị theo đường đi ngắn nhất có thể. Mặt khác, các nút A, B, & C có trung tâm nằm giữa bằng nhau, vì tất cả chúng đều nằm trên 1 số lượng lớn các đường đi ngắn nhất giữa các nút khác.
In addition to these measures, see how to calculate importance of a node using ML techniques in next chaps. However, it is not only measure covered. Indeed, *density* is another important measure, indicating how connected graph is. It is a ratio between actual number of edges & maximum possible number of edges in graph. A graph with high density is considered more connected & has more information flow compared to a graph with low density.
– Ngoài các phép đo này, xem cách tính tầm quan trọng của 1 nút bằng kỹ thuật ML trong các chương tiếp theo. Tuy nhiên, đây không chỉ là phép đo được đề cập. Thật vậy, *density* là 1 phép đo quan trọng khác, cho biết mức độ kết nối của đồ thị. Nó là tỷ lệ giữa số cạnh thực tế & số cạnh tối đa có thể có trong đồ thị. 1 đồ thị có mật độ cao được coi là kết nối hơn & có luồng thông tin lớn hơn so với đồ thị có mật độ thấp.
Formula to calculate density depends on whether graph is directed or undirected. For an undirected graph with n nodes, maximum possible number of edges is $\frac{n(n-1)}{2}$. For a directed graph with n nodes, maximum number of edges is $n(n-1)$. Density of a graph is calculated as number of edges divided by maximum number of edges.
– Công thức tính mật độ phụ thuộc vào việc đồ thị có hướng hay vô hướng. Đối với đồ thị vô hướng có n nút, số cạnh tối đa có thể là $\frac{n(n-1)}{2}$. Đối với đồ thị có hướng có n nút, số cạnh tối đa là $n(n-1)$. Mật độ của đồ thị được tính bằng số cạnh chia cho số cạnh tối đa.
A dense graph has a density closer to 1, while a sparse graph has a density closer to 0. There is no strict rule for what constitutes a dense or sparse graph, but generally, a graph is considered dense if its density is > 0.5 & sparse if its density is < 0.1. This measure is directly connected to a fundamental problem with graphs: how to represent adjacency matrix.
– Đồ thị dày đặc có mật độ gần bằng 1, trong khi đồ thị thưa có mật độ gần bằng 0. Không có quy tắc nghiêm ngặt nào về việc thế nào là đồ thị dày đặc hay thưa thớt, nhưng nhìn chung, 1 đồ thị được coi là dày đặc nếu mật độ của nó $>$ 0,5 & thưa nếu mật độ của nó $<$ 0,1. Phép đo này liên quan trực tiếp đến 1 vấn đề cơ bản với đồ thị: cách biểu diễn ma trận kề.

∗ Adjacency matrix representation. An adjacency matrix is a matrix that represents edges in a graph, where each cell indicates whether there is an edge between 2 nodes. Adjacency matrix is a square matrix of size $n \times n$, where n: number of nodes in graph. A value of 1 in cell (i, j) indicates that there is an edge between node i & node j, while a value of 0 indicates that there is no edge. For an undirected graph, adjacency matrix is symmetric, while for a directed graph, adjacency matrix is not necessarily symmetric. In Python, it can be implemented as a list of lists:
– Biểu diễn ma trận kề. Ma trận kề là 1 ma trận biểu diễn các cạnh trong đồ thị, trong đó mỗi ô biểu thị liệu có cạnh giữa

2 nút hay không. Ma trận kề là 1 ma trận vuông có kích thước $n \times n$, trong đó n: số nút trong đồ thị. Giá trị 1 trong ô (i, j) biểu thị rằng có 1 cạnh giữa nút i & nút j, trong khi giá trị 0 biểu thị rằng không có cạnh nào. Đối với đồ thị vô hướng, ma trận kề là đối xứng, trong khi đối với đồ thị có hướng, ma trận kề không nhất thiết phải đối xứng. Trong Python, ma trận kề có thể được triển khai dưới dạng danh sách các danh sách.
Adjacency matrix is a straightforward representation that can be easily visualized as a 2D array. 1 of key advantages of using an adjacency matrix: checking whether 2 nodes are connected is a constant time operation $O(1)$. This makes it an efficient way to test existence of an edge in graph. Moreover, it is used to perform matrix operations, which are useful for certain graph algorithms, e.g. calculating shortest path between 2 nodes.
– Ma trận kề là 1 biểu diễn đơn giản có thể dễ dàng hình dung dưới dạng 1 mảng 2D. 1 trong những lợi thế chính của việc sử dụng ma trận kề: việc kiểm tra xem 2 nút có được kết nối hay không là 1 phép toán thời gian hằng số $O(1)$. Điều này khiến nó trở thành 1 cách hiệu quả để kiểm tra sự tồn tại của 1 cạnh trong đồ thị. Hơn nữa, nó được sử dụng để thực hiện các phép toán ma trận, hữu ích cho 1 số thuật toán đồ thị, e.g.: tính toán đường đi ngắn nhất giữa 2 nút.
However, adding or removing nodes can be costly, as matrix needs to be resized or shifted. 1 of main drawbacks of using an adjacency matrix is its space complexity: as number of nodes in graph grows, space required to store adjacency matrix increases exponentially. Formally, say: adjacency matrix has a space complexity of $O(|V|^2)$, where $|V|$ represents number of nodes in graph.
– Tuy nhiên, việc thêm hoặc xóa các nút có thể tốn kém, vì ma trận cần được thay đổi kích thước hoặc dịch chuyển. 1 trong những nhược điểm chính của việc sử dụng ma trận kề là độ phức tạp về không gian: khi số lượng nút trong đồ thị tăng lên, không gian cần thiết để lưu trữ ma trận kề cũng tăng theo cấp số nhân. Ví dụ: ma trận kề có độ phức tạp về không gian là $O(|V|^2)$, trong đó $|V|$ biểu thị số lượng nút trong đồ thị.
Overall, while adjacency matrix is a useful data structure for representing small graphs, it may not be practical for larger ones due to its space complexity. Additionally, overhead of adding or removing nodes can make it inefficient for dynamically changing graphs.
– Nhìn chung, mặc dù ma trận kề là 1 cấu trúc dữ liệu hữu ích để biểu diễn các đồ thị nhỏ, nhưng nó có thể không thực tế đối với các đồ thị lớn hơn do độ phức tạp về không gian. Ngoài ra, việc thêm hoặc bớt các nút có thể khiến ma trận kề không hiệu quả đối với các đồ thị thay đổi động.
This is why other representation can be helpful. E.g., another popular way to store graphs is *edge list*. An edge list is a list of all edges in a graph. Each edge is represented by a tuple or a pair of vertices. Edge list can also include weight or cost of each edge. This is data structure used to create our graphs with `networkx`:
– Đây là lý do tại sao các biểu diễn khác có thể hữu ích. E.g., 1 cách phổ biến khác để lưu trữ đồ thị là `edge list`. Danh sách cạnh là danh sách tất cả các cạnh trong 1 đồ thị. Mỗi cạnh được biểu diễn bằng 1 bộ hoặc 1 cặp đỉnh. Danh sách cạnh cũng có thể bao gồm trọng số hoặc chi phí của mỗi cạnh. Đây là cấu trúc dữ liệu được sử dụng để tạo đồ thị của chúng ta với `networkx`:

```
edge_list = [(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)]
```

When compare both data structures applied to our graph, clear: edge list is less verbose. This is case because our graph is fairly sparse. On other hand, if our graph was complete, would require 21 tuples instead of 6. This is explained by a space complexity of $O(|E|)$, where $|E|$ is number of edges. Edge lists are more efficient for storing sparse graphs, where number of edges is much smaller than number of nodes.
– Khi so sánh cả 2 cấu trúc dữ liệu được áp dụng cho đồ thị của chúng ta, rõ ràng: danh sách cạnh ít chi tiết hơn. Điều này là do đồ thị của chúng ta khá thưa thớt. Mặt khác, nếu đồ thị của chúng ta đầy đủ, sẽ cần 21 bộ thay vì 6. Điều này được giải thích bởi độ phức tạp không gian là $O(|E|)$, trong đó $|E|$ là số cạnh. Danh sách cạnh hiệu quả hơn trong việc lưu trữ đồ thị thưa thớt, trong đó số cạnh nhỏ hơn nhiều so với số nút.
However, checking whether 2 vertices are connected in an edge list requires iterating through entire list, which can be time-consuming for large graphs with many edges. Therefore, edge lists are commonly used in applications where space is a concern.
– Tuy nhiên, việc kiểm tra xem 2 đỉnh có được kết nối trong danh sách cạnh hay không đòi hỏi phải lặp lại toàn bộ danh sách, điều này có thể tốn thời gian đối với các đồ thị lớn có nhiều cạnh. Do đó, danh sách cạnh thường được sử dụng trong các ứng dụng đòi hỏi không gian.
A 3rd & popular representation is *adjacency list*. It consists of a list of pairs, where each pair represents a node in graph & its adjacent nodes. The pairs can be stored in a linked list, dictionary, or other data structures, depending on implementation. An adjacency list has several advantages over an adjacency matrix or an edge list. 1st, space complexity is $O(|V| + |E|)$. This is more efficient than $O(|V|^2)$ space complexity of an adjacency matrix for sparse graphs. 2nd, it allows for efficient iteration through adjacent vertices of a node, which is useful in many graph algorithms. Finally, adding a node or an edge can be done in constant time.
– Biểu diễn phổ biến thứ 3 là *adjacency list*. Nó bao gồm 1 danh sách các cặp, trong đó mỗi cặp biểu diễn 1 nút trong đồ thị & các nút liền kề của nó. Các cặp có thể được lưu trữ trong danh sách liên kết, từ điển hoặc các cấu trúc dữ liệu khác, tùy thuộc vào cách triển khai. Danh sách kề có 1 số ưu điểm so với ma trận kề hoặc danh sách cạnh. Thứ nhất, độ phức tạp không gian là $O(|V| + |E|)$. Điều này hiệu quả hơn độ phức tạp không gian $O(|V|^2)$ của ma trận kề đối với đồ thị thưa. Thứ hai, nó cho phép lặp hiệu quả qua các đỉnh liền kề của 1 nút, điều này hữu ích trong nhiều thuật toán đồ thị. Cuối cùng, việc thêm 1 nút hoặc 1 cạnh có thể được thực hiện trong thời gian không đổi.
However, checking whether 2 vertices are connected can be slower than with an adjacency matrix. This is because it requires iterating through adjacency list of 1 of vertices, which can be time-consuming for large graphs.

– Tuy nhiên, việc kiểm tra xem 2 đỉnh có được kết nối hay không có thể chậm hơn so với ma trận kề. Điều này là do ma trận này yêu cầu phải lặp qua danh sách kề của 1 đỉnh, điều này có thể tốn thời gian đối với các đồ thị lớn.

Each data structure has its own advantages & disadvantages that depend on specific application & requirements. In next sect, process graphs & introduce 2 most fundamental graph algorithms.

– Mỗi cấu trúc dữ liệu đều có ưu điểm & nhược điểm riêng, tùy thuộc vào yêu cầu & ứng dụng cụ thể. Trong phần tiếp theo, đồ thị quy trình & giới thiệu 2 thuật toán đồ thị cơ bản nhất.

- Exploring graph algorithms. Graph algorithms are critical in solving problems related to graphs, e.g. finding shortest path between 2 nodes or detecting cycles. This sect will discuss 2 graph traversal algorithms: BFS & DFS.

  – Khám phá thuật toán đồ thị. Thuật toán đồ thị đóng vai trò quan trọng trong việc giải quyết các bài toán liên quan đến đồ thị, e.g.: tìm đường đi ngắn nhất giữa 2 nút hoặc phát hiện chu trình. Phần này sẽ thảo luận về 2 thuật toán duyệt đồ thị: BFS & DFS.

  * Breadth-first search. BFS is a graph traversal algorithm that starts at root node & explores all neighboring nodes at a particular level before moving to next level of nodes. It works by maintaining a queue of nodes to visit & marking each visited node as it is added to queue. Algorithm then dequeues next node in queue & explores all its neighbors, adding them to queue if they have not been visited yet. How can implement BFS in Python:
    · Create an empty graph & add edges with `add_edges_from()` method.
    · Define a function called `bfs()` that implements BFs algorithm on a graph. Function takes 2 arguments: `graph` object & starting node for search.
    · Initialize 2 lists `visited, queue` & add starting node. `visited` list keeps track of nodes that have been visited during search, while `queue` list stores nodes that need to be visited.
    · Enter a `while` loop that continues until `queue` list is empty. Inside loop, remove 1st node in `queue` list using `pop(0)` method & store result in `node` variable.
    · Iterate through neighbors of node using a `for` loop. For each neighbor that has not been visited yet, add it to `visited` list & to end of `queue` list using `append()` method. When it is complete, return `visited` list.
    · Call `bfs()` function with `G` argument & `'A'` starting node: `bfs(G, 'A')`.
    · Function returns list of visited nodes in order in which they were visited.

    BFS is particularly useful in finding shortest path between 2 nodes in an unweighted graph. This is because algorithm visits nodes in order of their distance from starting node, so 1st time target node is visited, it must be along shortest path from starting node.

    – BFS đặc biệt hữu ích trong việc tìm đường đi ngắn nhất giữa 2 nút trong đồ thị không trọng số. Điều này là do thuật toán sẽ duyệt các nút theo thứ tự khoảng cách từ nút bắt đầu, do đó, lần đầu tiên nút đích được duyệt, nó phải nằm trên đường đi ngắn nhất từ nút bắt đầu.

    In addition to finding shortest path, BFS can also be used to check whether a graph is connected or to find all connected components of a graph. It is also used in applications e.g. web crawlers, social network analysis, & shortest path routing in networks.

    – Ngoài việc tìm đường đi ngắn nhất, BFS còn có thể được sử dụng để kiểm tra xem 1 đồ thị có liên thông hay không hoặc để tìm tất cả các thành phần liên thông của đồ thị. Nó cũng được sử dụng trong các ứng dụng như trình thu thập dữ liệu web, phân tích mạng xã hội & định tuyến đường đi ngắn nhất trong mạng.

    Time complexity of BFS is $O(|V| + |E|)$. This can be a significant issue for graphs with a high degree of connectivity or for graphs that are sparse. Several variants of BFS have been developed to mitigate this issue, e.g. bidirectional BFS & A* search, wjich use heuristics to reduce number of nodes that need to be explored.

    – Độ phức tạp thời gian của BFS là $O(|V| + |E|)$. Điều này có thể là 1 vấn đề đáng kể đối với các đồ thị có mức độ kết nối cao hoặc các đồ thị thưa thớt. 1 số biến thể của BFS đã được phát triển để giảm thiểu vấn đề này, e.g.: tìm kiếm BFS song hướng & A*, sử dụng các thuật toán tìm kiếm để giảm số lượng nút cần khám phá.

  * Depth-first search. DFS is a recursive algorithm that starts at root node & explores as far as possible along each branch before backtracking. It chooses a node & explores all of its unvisited neighbors, visiting 1st neighbor that has not been explored & backtracking only when all neighbors have been visited. By doing so, it explores graph by following as deep a path from starting node as possible before backtracking to explore other branches. This continues until all nodes have been explored. Implement DFS in Python:
    1. 1st initialize an empty list called `visited`.
    2. Define a function called `dfs()` that takes in `visited, graph, node` as arguments: `def dfs(visited, graph, node):`
    3. If current `node` is not in `visited` list, append it to list.
    4. Then iterate through each neighbor of current `node`. For each neighbor, recursively call `dfs()` function passing in `visited, graph, neighbor` as arguments.
    5. dfs() function continues to explore graph depth-1st, visiting all neighbors of each node until there are no more unvisited neighbors. Finally, `visited` list is returned.

  – Tìm kiếm theo chiều sâu. DFS là 1 thuật toán đệ quy bắt đầu từ nút gốc & khám phá càng xa càng tốt dọc theo mỗi nhánh trước khi quay lui. Nó chọn 1 nút & khám phá tất cả các hàng xóm chưa được thăm của nó, thăm hàng xóm thứ nhất chưa được thăm & chỉ quay lui khi tất cả các hàng xóm đã được thăm. Bằng cách này, nó khám phá đồ thị bằng cách đi theo đường dẫn sâu nhất có thể từ nút bắt đầu trước khi quay lui để khám phá các nhánh khác. Quá trình này tiếp tục cho đến khi tất cả các nút đã được khám phá. Triển khai DFS trong Python:

1. Đầu tiên, khởi tạo 1 danh sách rỗng có tên là `visited`.
2. Định nghĩa 1 hàm có tên là `dfs()` nhận `visited, graph, node` làm đối số: `def dfs(visited, graph, node):`
3. Nếu `node` hiện tại không có trong danh sách `visited`, thêm nó vào danh sách.
4. Sau đó, lặp qua từng hàng xóm của `nút` hiện tại. Với mỗi hàng xóm, gọi đệ quy hàm `dfs()` truyền vào `visited, graph, neighbor` làm đối số.
5. Hàm dfs() tiếp tục khám phá đồ thị theo chiều sâu 1, duyệt tất cả các hàng xóm của mỗi nút cho đến khi không còn hàng xóm nào chưa được duyệt. Cuối cùng, danh sách `visited` được trả về.

DFS is useful in solving various problems, e.g. finding connected components, topological sorting, & solving maze problems. It is particularly useful in finding cycles in a graph since it traverses graph in a depth-1st order, & a cycle exists iff a node is visited twice during traversal.

– DFS hữu ích trong việc giải quyết nhiều bài toán khác nhau, e.g. tìm các thành phần liên thông, sắp xếp tôpô, & giải các bài toán mê cung. Nó đặc biệt hữu ích trong việc tìm chu trình trong đồ thị vì nó duyệt đồ thị theo thứ tự độ sâu 1, & tồn tại 1 chu trình chỉ khi & chỉ khi 1 nút được thăm 2 lần trong quá trình duyệt.

Like BFS, it has a time complexity of $O(|V| + |E|)$. It requires less memory but does not guarantee shallowest path solution. Finally, unlike BFS, you can be trapped in infinite loops using DFS.

– Giống như BFS, DFS có độ phức tạp thời gian là $O(|V| + |E|)$. Nó đòi hỏi ít bộ nhớ hơn nhưng không đảm bảo giải pháp đường đi nông nhất. Cuối cùng, không giống như BFS, bạn có thể bị mắc kẹt trong vòng lặp vô hạn khi sử dụng DFS.

Additionally, many other algorithms in graph theory build upon BFS & DFS, e.g. Dijkstra's shortest path algorithm, Kruskal's minimum spanning tree algorithm, & Tarjan's strongly connected components algorithm. Therefore, a solid understanding of BFS & DFS is essential for anyone who wants to work with graphs & develop more advanced graph algorithms.

– Ngoài ra, nhiều thuật toán khác trong lý thuyết đồ thị được xây dựng dựa trên BFS & DFS, e.g. thuật toán đường đi ngắn nhất Dijkstra, thuật toán cây bao trùm nhỏ nhất Kruskal, & thuật toán thành phần liên thông mạnh Tarjan. Do đó, việc hiểu rõ về BFS & DFS là điều cần thiết cho bất kỳ ai muốn làm việc với đồ thị & phát triển các thuật toán đồ thị nâng cao hơn.

- Summary. In this chap, covered essentials of graph theory, a branch of mathematics that studies graphs & networks. Began by defining what a graph is & explained different types of graphs, e.g., directed, weighted, & connected graphs. Then introduced fundamental graph objects (including neighbors) & measures (e.g. centrality & density), which are used to understand & analyze graph structures.

  – Chương này trình bày những kiến thức cơ bản về lý thuyết đồ thị, 1 nhánh của toán học nghiên cứu đồ thị & mạng lưới. Bắt đầu bằng việc định nghĩa đồ thị & giải thích các loại đồ thị khác nhau, e.g.: đồ thị có hướng, có trọng số, & liên thông. Sau đó, giới thiệu các đối tượng đồ thị cơ bản (bao gồm cả các lân cận) & các phép đo (e.g.: độ tập trung & mật độ), được sử dụng để hiểu & phân tích cấu trúc đồ thị.

  Additionally, discussed adjacency matrix & its different representations. Finally, explored 2 fundamental graph algorithms, BFS & DFS, which form foundation for developing more complex graph algorithms.

  – Ngoài ra, chúng tôi còn thảo luận về ma trận kề & các biểu diễn khác nhau của nó. Cuối cùng, chúng tôi đã khám phá 2 thuật toán đồ thị cơ bản, BFS & DFS, tạo thành nền tảng cho việc phát triển các thuật toán đồ thị phức tạp hơn.

  In Chap. 3: Creating Node Representation with DeepWalk, explore DeepWalk architecture & its 2 components: `Word2Vec` & random walks. Start by understanding `Word2Vec` architecture & then implement it using a specialized library. Then, delve into DeepWalk algorithm & implement random walks on a graph.

  – Trong Chương 3: Tạo Biểu diễn Nút với DeepWalk, khám phá kiến trúc DeepWalk & 2 thành phần của nó: `Word2Vec` & bước ngẫu nhiên. Bắt đầu bằng cách tìm hiểu kiến trúc `Word2Vec` & sau đó triển khai nó bằng 1 thư viện chuyên dụng. Sau đó, đi sâu vào thuật toán DeepWalk & triển khai bước ngẫu nhiên trên đồ thị.

- 3. Creating Node Representations with DeepWalk. DeepWalk is 1 of 1st major successful applications of ML techniques to graph data. It introduces important concepts e.g. embeddings that are at core of GNNs. Unlike traditional neural networks, goal of this architecture: produce representations that are then fed to other models, which perform downstream tasks (e.g., node classification).

  – 3. Tạo Biểu diễn Nút với DeepWalk. DeepWalk là 1 trong những ứng dụng thành công đầu tiên của kỹ thuật ML vào đồ thị dữ liệu. Nó giới thiệu các khái niệm quan trọng, e.g. nhúng, vốn là cốt lõi của mạng nơ-ron nhân tạo (GNN). Không giống như mạng nơ-ron truyền thống, mục tiêu của kiến trúc này là tạo ra các biểu diễn sau đó được đưa vào các mô hình khác, thực hiện các tác vụ tiếp theo (e.g.: phân loại nút).

  In this chap, learn about DeepWalk architecture & its 2 major components: Word2Vec & random walks. Explain how Word2Vec architecture works, with a particular focus on skip-gram model. Implement this model with popular `gensim` library on a NLP example to understand how it is supposed to be used.

  – Trong chương này, chúng ta sẽ tìm hiểu về kiến trúc DeepWalk & 2 thành phần chính của nó: Word2Vec & random walks. Giải thích cách thức hoạt động của kiến trúc Word2Vec, đặc biệt tập trung vào mô hình skip-gram. Triển khai mô hình này với thư viện `gensim` phổ biến trên 1 ví dụ NLP để hiểu cách sử dụng.

  Then focus on DeepWalk algorithm & see how performance can be improved using hierarchical softmax (H-Softmax). This powerful optimization of softmax function can be found in many fields: it is incredibly useful when you have a lot of possible

classes in your classification task. Also implement random walks on a graph before wrapping things up with an end-to-end supervised classification exercise on Zachary's Karate Club.

– Sau đó, tập trung vào thuật toán DeepWalk & xem cách cải thiện hiệu suất bằng cách sử dụng softmax phân cấp (H-Softmax). Việc tối ưu hóa mạnh mẽ hàm softmax này có thể được tìm thấy trong nhiều lĩnh vực: nó cực kỳ hữu ích khi bạn có nhiều lớp khả thi trong tác vụ phân loại của mình. Hãy triển khai các bước ngẫu nhiên trên đồ thị trước khi kết thúc bằng bài tập phân loại có giám sát đầu cuối trên Zachary's Karate Club.

By end of this chap, master Word2Vec in context of NLP & beyond. Able to create node embeddings using topological information of graphs & solve classification tasks on graph data. In this chap, cover main topics: introducing Word2Vec, DeepWalk & random walks, implementing DeepWalk.

– Đến cuối chương này, bạn sẽ nắm vững Word2Vec trong ngữ cảnh NLP & hơn thế nữa. Có khả năng tạo nhúng nút bằng thông tin tôpô của đồ thị & giải quyết các bài toán phân loại trên dữ liệu đồ thị. Trong chương này, bạn sẽ tìm hiểu các chủ đề chính: giới thiệu Word2Vec, DeepWalk & bước ngẫu nhiên, & triển khai DeepWalk.

- Introducing Word2Vec. 1st step to comprehending DeepWalk algorithm: understand its major component: Word2Vec. Word2Vec has been 1 of most influential DL techniques in NLP. Published in 2013 by Tomas Mikolov et al. (Google) in 2 different papers, it proposed a new technique to translate words into vectors (also known as *embeddings*) using large datasets of text. These representations can then be used in downstream tasks, e.g. sentiment classification. It is also 1 of rare examples of patented & popular ML architecture.

  – Giới thiệu Word2Vec. Bước đầu tiên để hiểu thuật toán DeepWalk: hiểu thành phần chính của nó: Word2Vec. Word2Vec là 1 trong những kỹ thuật DL có ảnh hưởng nhất trong NLP. Được công bố vào năm 2013 bởi Tomas Mikolov & cộng sự (Google) trong 2 bài báo khác nhau, kỹ thuật này đã đề xuất 1 kỹ thuật mới để dịch từ thành các vectơ (còn được gọi là nhúng) bằng cách sử dụng các tập dữ liệu văn bản lớn. Các biểu diễn này sau đó có thể được sử dụng trong các tác vụ tiếp theo, e.g. phân loại cảm xúc. Đây cũng là 1 trong những ví dụ hiếm hoi về kiến trúc ML phổ biến & đã được cấp bằng sáng chế.

  A few examples of how Word2Vec can transform words into vectors: $vec(king) = [-2.1, 4.1, 0.6], vec(queen) = [-1.9, 2.6, 1.5], vec(ma$ $[3.0, -1.1, -2], vec(woman) = [2.8, -2.6, -1.1]$. Can see in this example: in terms of Euclidean distance, word vectors for *king & queen* are closer than ones for *king & woman* (4.37 vs. 8.47). In general, other metrics, e.g. popular *cosine similarity*, are used to measure likeness of these words. Cosine similarity focuses on angle between vectors & does not consider their magnitude (length), which is more helpful in comparing them. Here is how it is defined:

$$\text{cosine similarity}(\vec{A}, \vec{B}) = \cos\theta = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|}.$$

  1 of most surprising results of Word2Vec is its ability to solve analogies. A popular example is how it can answer question "man is to woman, what kind is to ...?" It can be calculated as follows: $vec(king) - vec(man) + vec(woman) \approx vec(queen)$. This is not true with any analogy, but this property can bring interesting applications to perform arithmetic operations with embeddings.

  – 1 vài ví dụ về cách Word2Vec có thể chuyển đổi từ thành vectơ: $vec(king) = [-2.1, 4.1, 0.6], vec(queen) = [-1.9, 2.6, 1.5], vec(man)$ $[3.0, -1.1, -2], vec(woman) = [2.8, -2.6, -1.1]$. Có thể thấy trong ví dụ này: về khoảng cách Euclidean, vectơ từ cho *king & queen* gần hơn vectơ cho *king & woman* (4,37 so với 8,47). Nhìn chung, các số liệu khác, e.g. *cosine similarity* phổ biến, được sử dụng để đo mức độ giống nhau của các từ này. Độ giống nhau cosine tập trung vào góc giữa các vectơ & không xem xét độ lớn (chiều dài) của chúng, điều này hữu ích hơn khi so sánh chúng. Đây là cách định nghĩa của nó:

$$\text{cosine similarity}(\vec{A}, \vec{B}) = \cos\theta = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|}.$$

  1 trong những kết quả đáng ngạc nhiên nhất của Word2Vec là khả năng giải quyết các phép loại suy. 1 ví dụ phổ biến là cách nó có thể trả lời câu hỏi "đàn ông là gì với phụ nữ, vậy loại nào là ...?". Nó có thể được tính như sau: $vec(king) - vec(man) + vec(woman) \approx vec(queen)$. Điều này không đúng với bất kỳ phép loại suy nào, nhưng tính chất này có thể mang lại những ứng dụng thú vị để thực hiện các phép toán số học với phép nhúng.

  * CBOW vs. skip-gram. A model must be trained on a pretext task to produce these vectors. Task itself does not need to be meaningful: its only goal is to produce high-quality embeddings. In practice, this task is always related to predicting words given a certain context.

    – CBOW so với skip-gram. 1 mô hình phải được huấn luyện trên 1 tác vụ giả định để tạo ra các vectơ này. Bản thân tác vụ không cần phải có ý nghĩa: mục tiêu duy nhất của nó là tạo ra các nhúng chất lượng cao. Trên thực tế, tác vụ này luôn liên quan đến việc dự đoán các từ trong 1 ngữ cảnh nhất định.

    Authors proposed 2 architectures with similar tasks:

    · **Continuous bag-of-words (CBOW) model.** This is trained to predict a word its surrounding context (words coming before & after target word). Order of context words does not matter since their embeddings are summed in model. Authors claim to obtain better results using 4 words before & after the one that is predicted.

    · **Continuous skip-gram model.** Here we feed a single word to model & try to predict words around it. Increasing range of context words lead to better embeddings but also increases training time.

In summary, here are inputs & outputs of both models: Fig. 3.1: CBOW & skip-gram architectures.

– Các tác giả đã đề xuất 2 kiến trúc với các nhiệm vụ tương tự:

· **Mô hình túi từ liên tục (CBOW).** Mô hình này được huấn luyện để dự đoán 1 từ trong ngữ cảnh xung quanh nó (các từ đứng trước & sau từ mục tiêu). Thứ tự của các từ ngữ cảnh không quan trọng vì các nhúng của chúng được tổng hợp trong mô hình. Các tác giả tuyên bố thu được kết quả tốt hơn khi sử dụng 4 từ đứng trước & sau từ được dự đoán.

· **Mô hình bỏ qua liên tục.** Ở đây, chúng tôi đưa 1 từ duy nhất vào mô hình & cố gắng dự đoán các từ xung quanh nó. Việc tăng phạm vi từ ngữ cảnh dẫn đến các nhúng tốt hơn nhưng cũng làm tăng thời gian huấn luyện.

Tóm lại, đây là đầu vào & đầu ra của cả 2 mô hình: Hình 3.1: Kiến trúc CBOW & bỏ qua.

In general, CBOW model is considered faster to train, but skip-gram model is more accurate thanks to its ability to learn infrequent words. This topic is still debated in NLP community: a different implementation could fix issues related to CBOW in some contexts.

– Nhìn chung, mô hình CBOW được đánh giá là nhanh hơn khi huấn luyện, nhưng mô hình skip-gram lại chính xác hơn nhờ khả năng học các từ ít phổ biến. Chủ đề này vẫn đang được tranh luận trong cộng đồng NLP: 1 triển khai khác có thể khắc phục các vấn đề liên quan đến CBOW trong 1 số bối cảnh.

∗ Creating skip-grams. For now, focus on skip-gram model since it is architecture used by DeepWalk. Skip-grams are implemented as pairs of words with following structure (target word, context word), where *target word* is input & *context word* is word to predict. Number of skip grams for same target word depends on a parameter called *context size*, as shown in Fig. 3.2: Text to skip-grams. Same idea can be applied to a corpus of text instead of a single sentence.

– Tạo skip-gram. Trước mắt, hãy tập trung vào mô hình skip-gram vì đây là kiến trúc được DeepWalk sử dụng. Skip-gram được triển khai theo cặp từ với cấu trúc sau (từ đích, từ ngữ cảnh), trong đó từ đích là đầu vào & từ ngữ cảnh là từ cần dự đoán. Số lượng skip-gram cho cùng 1 từ đích phụ thuộc vào 1 tham số gọi là kích thước ngữ cảnh, như thể hiện trong Hình 3.2: Văn bản thành skip-gram. Ý tưởng tương tự có thể được áp dụng cho 1 tập hợp văn bản thay vì 1 câu đơn lẻ.

In practice, store all context words for same target word in a list to save memory. See how it is done with an example on an entire paragraph. In following example, create skip-grams for an entire paragraph stored in `text` variable. Set `CONTEXT_SIZE` variable to 2, i.e., look at 2 words before & after our target word:

1. Start by importing necessary libraries: numpy.
2. Then need to set `CONTEXT_SIZE` variable to 2 & bring in text we want to analyze:
3. Next create skip-grams thanks to a simple `for` loop to consider every word in `text`. A list comprehension generates context words, stored in `skipgrams` list:

```
skipgrams = []
for i in range(CONTEXT_SIZE, len(text) - CONTEXT_SIZE):
    array = [text[j] for j in np.arange(i - CONTEXT_SIZE, i + CONTEXT_SIZE + 1) if j != i]
    skipgrams.append((text[i], array))
```

4. Finally, use `print()` function to see skip-grams we generated:

```
print(skipgrams[0:2])
```

These 2 target words, with their corresponding context, work to show what inputs to Word2Vec look like.

– Trong thực tế, hãy lưu trữ tất cả các từ ngữ cảnh cho cùng 1 từ mục tiêu trong 1 danh sách để tiết kiệm bộ nhớ. Xem cách thực hiện với ví dụ trên toàn bộ 1 đoạn văn. Trong ví dụ sau, hãy tạo skip-gram cho toàn bộ 1 đoạn văn được lưu trữ trong biến `text`. Đặt biến `CONTEXT_SIZE` thành 2, i.e., xem xét 2 từ trước & sau từ mục tiêu của chúng ta:

1. Bắt đầu bằng cách nhập các thư viện cần thiết: numpy.
2. Sau đó, cần đặt biến `CONTEXT_SIZE` thành 2 & nhập văn bản chúng ta muốn phân tích:
3. Tiếp theo, hãy tạo skip-gram nhờ vòng lặp `for` đơn giản để xem xét mọi từ trong `text`. 1 danh sách hiểu biết tạo ra các từ ngữ cảnh, được lưu trữ trong danh sách `skipgrams`:

```
skipgrams = []
for i in range(CONTEXT_SIZE, len(text) - CONTEXT_SIZE):
    array = [text[j] for j in np.arange(i - CONTEXT_SIZE, i + CONTEXT_SIZE + 1) if j != i]
    skipgrams.append((text[i], array))
```

4. Cuối cùng, sử dụng hàm `print()` để xem các skip-gram mà chúng ta đã tạo ra:

```
print(skipgrams[0:2])
```

Hai từ mục tiêu này, cùng với ngữ cảnh tương ứng, sẽ hiển thị các dữ liệu đầu vào của Word2Vec trông như thế nào.

∗ skip-gram model. Goal of Word2Vec: produce high-quality word embeddings. To learn these embeddings, training task of skip-gram model consists of predicting correct context words given a target word.

– mô hình skip-gram. Mục tiêu của Word2Vec: tạo ra các nhúng từ chất lượng cao. Để học các nhúng này, nhiệm vụ huấn luyện của mô hình skip-gram bao gồm dự đoán các từ ngữ cảnh chính xác dựa trên 1 từ mục tiêu.

Imagine we have a sequence of N words $w_1, w_2, \ldots, w_N$. Probability of seeing word w_2 given word w_1 is written $p(w_2|w_1)$. Goal: maximize sum of every probability of seeing a context word given a target word in an entire text:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{-c\leq j\leq c,\ j\neq 0}\log p(w_{n+j}|w_n),$$

where c is size of context vector.
– Tưởng tượng chúng ta có 1 chuỗi N từ $w_1, w_2, \ldots, w_N$. Xác suất nhìn thấy từ w_2 cho từ w_1 được viết là $p(w_2|w_1)$. Mục tiêu: tối đa hóa tổng của mọi xác suất nhìn thấy 1 từ ngữ cảnh cho 1 từ đích trong toàn bộ văn bản:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{-c\leq j\leq c,\ j\neq 0}\log p(w_{n+j}|w_n),$$

trong đó c là kích thước của vectơ ngữ cảnh.

Remark 1. *Why do we use a log probability in prev equation? Transforming probabilities into log probabilities is a common technique in ML (& CS in general) for 2 main reasons. Products become additions (& divisions become subtractions). Multiplications are more computationally expensive than additions, so it is faster to compute log probability:* $\log(ab) = \log a + \log b$. *Way computers store very small numbers, e.g.,* `3.14e-128` *is not perfectly accurate, unlike log of same numbers* -127.5 *in this case. These small errors can add up & bias final results when events are extremely unlikely. On whole, this simple transformation allows us to gain speed & accuracy without changing our initial objective.*
– Tại sao chúng ta sử dụng xác suất logarit trong phương trình prev? Việc chuyển đổi xác suất thành xác suất logarit là 1 kỹ thuật phổ biến trong ML (& CS nói chung) vì 2 lý do chính. Tích trở thành phép cộng (& phép chia trở thành phép trừ). Phép nhân tốn kém về mặt tính toán hơn phép cộng, do đó, tính xác suất logarit nhanh hơn: $\log(ab) = \log a + \log b$. *Cách máy tính lưu trữ những số rất nhỏ, e.g.:* `3.14e-128` *không hoàn toàn chính xác, không giống như logarit của cùng 1 số* -127.5 *trong trường hợp này. Những sai số nhỏ này có thể cộng lại & làm sai lệch kết quả cuối cùng khi các sự kiện cực kỳ khó xảy ra. Nhìn chung, phép biến đổi đơn giản này cho phép chúng ta tăng tốc & độ chính xác mà không làm thay đổi mục tiêu ban đầu.*

Basic skip-gram models uses softmax function to calculate probability of a context word embedding h_c given a target word embedding h_t:

$$p(w_c|w_t) = \frac{\exp(h_c h_t^\top)}{\sum_{i=1}^{|V|}\exp(h_i h_t^\top)},$$

where V is vocabulary of size $|V|$. This vocabulary corresponds to list of unique words the model tries to predict. Can obtain this list using `set` data structure to remove duplicate words:
– Các mô hình skip-gram cơ bản sử dụng hàm softmax để tính xác suất nhúng từ ngữ cảnh h_c cho 1 từ mục tiêu nhúng h_t:

$$p(w_c|w_t) = \frac{\exp(h_c h_t^\top)}{\sum_{i=1}^{|V|}\exp(h_i h_t^\top)},$$

trong đó V là từ vựng có kích thước $|V|$. Từ vựng này tương ứng với danh sách các từ duy nhất mà mô hình cố gắng dự đoán. Có thể lấy danh sách này bằng cách sử dụng cấu trúc dữ liệu `set` để loại bỏ các từ trùng lặp:

```
# skip-gram model
vocab = set(text)
VOCAB_SIZE = len(vocab)
print(f"Length of vocabulary = {VOCAB_SIZE}")
```

Now have size of our vocabulary, there is 1 more parameter we need to define: N, dimensionality of word vectors. Typically, this value is set between 100 & 1000. In this example, set it to 10 because of limited size of our dataset.
– Bây giờ, với kích thước của vốn từ vựng, chúng ta cần định nghĩa thêm 1 tham số nữa: N, số chiều của vectơ từ. Thông thường, giá trị này được đặt trong khoảng từ 100 đến 1000. Trong ví dụ này, hãy đặt thành 10 vì kích thước tập dữ liệu của chúng ta có hạn.
Skip-gram model is composed of only 2 layers:

- A *projection layer* with a weight matrix W_{embed}, which takes a 1-hot encoded-word vector as an input & returns corresponding N-dim word embedding. It acts as a simple lookup table that stores embeddings of a predefined dimensionality.
- A *fully connected layer* with a weight matrix W_{output}, which takes a word embedding as input & outputs $|V|$-dim logits. A softmax function is applied to these predictions to transform logits into probabilities.

– Mô hình Skip-gram chỉ bao gồm 2 lớp:

- A *projection layer* với ma trận trọng số W_{embed}, lấy 1 vectơ từ được mã hóa 1-hot làm đầu vào & trả về nhúng từ N-dim tương ứng. Nó hoạt động như 1 bảng tra cứu đơn giản lưu trữ các nhúng có chiều được xác định trước.
- A *fully connected layer* với ma trận trọng số W_{output}, lấy 1 nhúng từ làm đầu vào & trả về logit $|V|$-dim. 1 hàm softmax được áp dụng cho các dự đoán này để chuyển đổi logit thành xác suất.

Remark 2. *There is no activation function:* `Word2Vec` *is a linear classifier that models a linear relationship between words.*

– Không có hàm kích hoạt: `Word2Vec` *là 1 bộ phân loại tuyến tính mô hình hóa mối quan hệ tuyến tính giữa các từ.*

Call $\mathbf{x}$ 1-hot encoded-word vector *input*. Corresponding word embedding can be calculated as a simple projection $h = W_{\text{embed}}^{\top} \cdot \mathbf{x}$. Using skip-gram model, can rewrite previous probability as follows:

$$p(w_c|w_t)\frac{\exp(W_{\text{output}} \cdot h)}{\sum_{i=1}^{|V|}\exp(W_{\text{output(i)}} \cdot h)}.$$

Skip-gram model outputs a $|V|$-dim vector, which is conditional probability of every word in vocabulary:

$$word2vec(w_t) = \begin{bmatrix} p(w_1|w_t) \\ p(w_2|w_t) \\ \vdots \\ p(w_{|V|}|w_t) \end{bmatrix}.$$

During training, these probabilities are compared to correct 1-hot encoded-target word vectors. Difference between these values (calculated by a loss function e.g. cross-entropy loss) is backpropagated through network to update weights & obtain better predictions.

– Gọi $\mathbf{x}$ là vectơ từ mã hóa 1-hot *input*. Nhúng từ tương ứng có thể được tính toán như 1 phép chiếu đơn giản $h = W_{\text{embed}}^{\top} \cdot \mathbf{x}$. Sử dụng mô hình skip-gram, có thể viết lại xác suất trước đó như sau:

$$p(w_c|w_t)\frac{\exp(W_{\text{output}} \cdot h)}{\sum_{i=1}^{|V|}\exp(W_{\text{output(i)}} \cdot h)}.$$

Mô hình Skip-gram đưa ra 1 vectơ $|V|$-dim, là xác suất có điều kiện của mọi từ trong từ vựng:

$$word2vec(w_t) = \begin{bmatrix} p(w_1|w_t) \\ p(w_2|w_t) \\ vdots \\ p(w_{|V|}|w_t) \end{bmatrix}.$$

Trong quá trình huấn luyện, các xác suất này được so sánh với các vectơ từ mục tiêu được mã hóa 1-hot chính xác. Sự khác biệt giữa các giá trị này (được tính bằng hàm mất mát, e.g.: mất mát entropy chéo) được truyền ngược qua mạng để cập nhật trọng số & thu được dự đoán tốt hơn.

Entire Word2Vec architecture is summarized in following diagram, with both matrices & final softmax layer: Fig. 3.3: Word2Vec architecture. Can implement this model using `gensim` library, which is also used in official implementation of DeepWalk. Can then build vocabulary & train our model based on previous text:

1. Begin by installing `gensim` & importing `Word2Vec` class:
2. Initialize a skip-gram model with a `Word2Vec` object & an `sg = 1` parameter (skip-gram = 1).
3. A good idea to check shape of our 1st weight matrix. It should correspond to vocabulary size & word embeddings' dimensionality.
4. Train model for 10 epochs.
5. Can print a word embedding to see what result of this training looks like.

While this approach works well with small vocabularies, computational cost of applying a full softmax function to millions of words (vocabulary size) is too costly in most cases. This has been a limiting factor in developing accurate language models for a long time. Fortunately for us, other approaches have been designed to solve this issue.

– Toàn bộ kiến trúc Word2Vec được tóm tắt trong sơ đồ sau, với cả ma trận & lớp softmax cuối cùng: Hình 3.3: Kiến trúc Word2Vec. Có thể triển khai mô hình này bằng thư viện `gensim`, cũng được sử dụng trong triển khai chính thức của DeepWalk. Sau đó, có thể xây dựng từ vựng & huấn luyện mô hình của chúng ta dựa trên văn bản trước đó:

1. Bắt đầu bằng cách cài đặt `gensim` & nhập lớp `Word2Vec`:
2. Khởi tạo mô hình skip-gram với đối tượng `Word2Vec` & tham số `sg = 1` (skip-gram = 1).
3. Nên kiểm tra hình dạng của ma trận trọng số thứ nhất. Nó phải tương ứng với kích thước từ vựng & chiều của nhúng từ.
4. Huấn luyện mô hình trong 10 kỷ nguyên.
5. Có thể in 1 nhúng từ để xem kết quả của quá trình huấn luyện này trông như thế nào.

Mặc dù phương pháp này hiệu quả với các từ vựng nhỏ, nhưng chi phí tính toán khi áp dụng hàm softmax đầy đủ cho hàng triệu từ (kích thước từ vựng) thường quá tốn kém. Đây là 1 yếu tố hạn chế trong việc phát triển các mô hình ngôn ngữ chính xác trong 1 thời gian dài. May mắn thay, các phương pháp khác đã được thiết kế để giải quyết vấn đề này.

Word2Vec (& DeepWalk) implements 1 of these techniques, called H-Softmax. Instead of a flat softmax that directly calculates probability of every word, this technique uses a binary tree structure where leaves are words. Even more interestingly, a Huffman tree can be used, where infrequent words are stored at deeper levels than common words. In most cases, this dramatically speeds up word prediction by a factor ≥ 50.

– Word2Vec (& DeepWalk) triển khai 1 trong những kỹ thuật này, được gọi là H-Softmax. Thay vì 1 softmax phẳng tính toán trực tiếp xác suất của từng từ, kỹ thuật này sử dụng cấu trúc cây nhị phân với các lá là các từ. Thú vị hơn nữa, có thể sử dụng cây Huffman, trong đó các từ ít phổ biến được lưu trữ ở các cấp độ sâu hơn so với các từ phổ biến. Trong hầu hết các trường hợp, điều này giúp tăng tốc đáng kể việc dự đoán từ lên gấp ≥ 50.

H-Softmax can be activated in `gensim` using `hs = 1`. This was most difficult part of DeepWalk architecture. But before can implement it, need 1 more component: how to create our training data.

– H-Softmax có thể được kích hoạt trong `gensim` bằng cách sử dụng `hs = 1`. Đây là phần khó nhất của kiến trúc DeepWalk. Nhưng trước khi có thể triển khai nó, cần thêm 1 thành phần nữa: cách tạo dữ liệu huấn luyện.

- DeepWalk & random walks. Proposed in 2014 by Perozzi et al., DeepWalk quickly became extremely popular among graph researchers. Inspired by recent advances in NLP, it consistently outperformed other methods on several datasets. While more performant architectures have been proposed since then, DeepWalk is a simple & reliable baseline that can be quickly implemented to solve a lot of problems.

  – DeepWalk & random walks. Được đề xuất vào năm 2014 bởi Perozzi & cộng sự, DeepWalk nhanh chóng trở nên cực kỳ phổ biến trong giới nghiên cứu đồ thị. Lấy cảm hứng từ những tiến bộ gần đây trong NLP, nó luôn vượt trội hơn các phương pháp khác trên 1 số tập dữ liệu. Mặc dù các kiến trúc hiệu suất cao hơn đã được đề xuất kể từ đó, DeepWalk là 1 đường cơ sở đơn giản & đáng tin cậy, có thể được triển khai nhanh chóng để giải quyết nhiều vấn đề.

  Goal of DeepWalk: produce high-quality feature representations of nodes in an unsupervised way. This architecture is heavily inspired by Word2Vec in NLP. However, instead of words, our dataset is composed of nodes. This is why we use random walks to generate meaningful sequence of nodes that act like sentences. Following diagram illustrates connection between sentences & graphs: Fig. 3.4: Sentences can be represented as graphs. Random walks are sequences of nodes produced by randomly choosing a neighboring node at every step. Thus, nodes can appear several times in same sequence.

  – Mục tiêu của DeepWalk: tạo ra các biểu diễn đặc trưng chất lượng cao của các nút theo cách không giám sát. Kiến trúc này lấy cảm hứng rất nhiều từ Word2Vec trong NLP. Tuy nhiên, thay vì các từ, tập dữ liệu của chúng tôi được tạo thành từ các nút. Đây là lý do tại sao chúng tôi sử dụng các bước ngẫu nhiên để tạo ra chuỗi các nút có ý nghĩa hoạt động như các câu. Sơ đồ sau minh họa mối liên hệ giữa các câu & đồ thị: Hình 3.4: Các câu có thể được biểu diễn dưới dạng đồ thị. Các bước ngẫu nhiên là các chuỗi nút được tạo ra bằng cách chọn ngẫu nhiên 1 nút lân cận ở mỗi bước. Do đó, các nút có thể xuất hiện nhiều lần trong cùng 1 chuỗi.

  Why are random walks important? Even if nodes are randomly selected, fact they often appear together in a sequence means: they are close to each other. Under *network homophily* hypothesis, notes that are close to each other are similar. This is particularly case in social networks, where people are connected to friends & family.

  – Tại sao các bước ngẫu nhiên lại quan trọng? Ngay cả khi các nút được chọn ngẫu nhiên, việc chúng thường xuất hiện cùng nhau theo 1 trình tự có nghĩa là: chúng gần nhau. Theo giả thuyết *network homophily*, các nốt gần nhau thì tương tự nhau. Điều này đặc biệt đúng trong các mạng xã hội, nơi mọi người kết nối với bạn bè & gia đình.

  This idea is at core of DeepWalk algorithm: when nodes are close to each other, want to obtain high similarity scores. On contrary, want low scores when they are farther apart. Implement a random walk function using a `networkx` graph:

  – Ý tưởng này là cốt lõi của thuật toán DeepWalk: khi các nút gần nhau, chúng ta muốn đạt được điểm tương đồng cao. Ngược lại, chúng ta muốn đạt điểm tương đồng thấp khi chúng ở xa nhau. Triển khai hàm bước ngẫu nhiên bằng đồ thị `networkx`:

  1. Import required libraries & initialize random number generator for reproducibility:
     ```
     mport networkx as nx
     import matplotlib.pyplot as plt
     import numpy as np
     import random
     random.seed(0) # initialize random number generator for reproducibility
     ```
  2. Generate a random graph thanks to `erdos_renyi_graph` function with a fixed number of nodes 10 & a predefined probability of creating an edge between 2 nodes 0.3:
     ```
     G = nx.erdos_renyi_graph(10, 0.3, seed = 1, directed = False)
     ```
  3. Plot this random graph to see what it looks like.
  4. Implement random walks with a simple function. This function takes 2 parameters: starting node `start` & length of walk `length`. At every step, randomly select a neighboring node (using `np.random.choice`) until walk is complete:
     ```
     # implement random walks
     def random_walk(start, length):
         walk = [str(start)] # starting node
         for i in range(length):
             neighbors = [node for node in G.neighbors(start)]
             next_node = np.random.choice(neighbors, 1)[0]
             walk.append(str(next_node))
             start = next_node
         return walk
     ```

5. Print result of this function with starting node as 0 & a length of 10:
   ```
   print(random_walk(0, 10))
   ```

Can see: certain nodes, e.g. 0 & 9, are often found together. Considering that it is a homophilic graph, i.e., they are similar. It is precisely type of relationship we are trying to capture with DeepWalk. Now have implemented Word2Vec & random walks separately, combine them to create DeepWalk.

– Có thể thấy: 1 số nút nhất định, e.g.: 0 & 9, thường được tìm thấy cùng nhau. Xét đến việc đây là 1 đồ thị đồng dạng, i.e., chúng tương tự nhau. Đây chính xác là loại mối quan hệ mà chúng ta đang cố gắng nắm bắt bằng DeepWalk. Bây giờ, chúng ta đã triển khai Word2Vec & random walks riêng biệt, hãy kết hợp chúng để tạo ra DeepWalk.

- Implementing DeepWalk. Now have a good understanding of every component in this architecture, use it to solve an ML problem. Dataset we use is Zachary's Karate Club. It simply represents relationships within a karate club studied by Wayne W. Zachary in 1970s. It is a kind of social network where every node is a member, & members who interact outside club are connected.

  – Triển khai DeepWalk. Bây giờ, hãy hiểu rõ từng thành phần trong kiến trúc này, hãy sử dụng nó để giải quyết 1 bài toán ML. Tập dữ liệu chúng tôi sử dụng là Câu lạc bộ Karate của Zachary. Nó đơn giản biểu diễn các mối quan hệ trong 1 câu lạc bộ karate được Wayne W. Zachary nghiên cứu vào những năm 1970. Đây là 1 dạng mạng xã hội, trong đó mỗi nút là 1 thành viên, & các thành viên tương tác bên ngoài câu lạc bộ được kết nối.

  In this example, club is divided into 2 groups: could like to assign right group to every member (node classification) just by looking at their connections:

  1. Import dataset using `nx.karate_club_graph()`:
     ```
     G = nx.karate_club_graph()
     ```
  2. Need to convert string class labels into numerical values `Mr. Hi = 0, Officer = 1`:
  3. Plot this graph using our new labels.
  4. Next step: generate our dataset, random walks. Want to be as exhaustive as possible, which is why we will create 80 random walks of a length of 10 for every node in graph.
  5. Print a walk to verify that it is correct: 1st walk that was generated.
  6. Final step consists of implementing Word2Vec. Here use skip-gram model previously seen with H-Softmax. Can play with other parameters to improve quality of embeddings.
  7. Model is then simply trained on random walks generated:
     ```
     model.train(walks, total_examples = model.corpus_count, epochs = 30, report_delay = 1)
     ```
  8. Now our model is trained, see its different applications. 1st one allows us to find most similar nodes to a given one (in terms of cosine similarity):
     ```
     print('Nodes that are the most similar to node 0:')
     for similarity in model.wv.most_similar(positive = ['0']):
         print(f' {similarity}')
     ```

     This produces following output for `Nodes` most similar to node 0. Another important application is calculating similarity score between 2 nodes. It can be performed as follows:
     ```
     # similarity between 2 nodes
     print(f"Similarity between node 0 & 4: {model.wv.similarity('0', '4')}")
     ```

– Trong ví dụ này, câu lạc bộ được chia thành 2 nhóm: có thể muốn gán đúng nhóm cho từng thành viên (phân loại nút) chỉ bằng cách xem xét các kết nối của họ:

1. Nhập tập dữ liệu bằng `nx.karate_club_graph()`:
   ```
   G = nx.karate_club_graph()
   ```
2. Cần chuyển đổi nhãn lớp chuỗi thành giá trị số `Mr. Hi = 0, Officer = 1`:
3. Vẽ đồ thị này bằng các nhãn mới của chúng ta.
4. Bước tiếp theo: tạo tập dữ liệu của chúng ta, các bước đi ngẫu nhiên. Muốn càng đầy đủ càng tốt, đó là lý do tại sao chúng ta sẽ tạo 80 bước đi ngẫu nhiên có độ dài 10 cho mỗi nút trong đồ thị.
5. In 1 bước đi để xác minh rằng nó chính xác: Bước đi đầu tiên đã được tạo.
6. Bước cuối cùng bao gồm việc triển khai Word2Vec. Ở đây sử dụng mô hình skip-gram đã thấy trước đó với H-Softmax. Có thể sử dụng các tham số khác để cải thiện chất lượng nhúng.
7. Mô hình sau đó được huấn luyện đơn giản dựa trên các bước đi ngẫu nhiên được tạo ra:
   ```
   model.train(walks, total_examples = model.corpus_count, epochs = 30, report_delay = 1)
   ```

8. Bây giờ mô hình của chúng ta đã được huấn luyện, hãy xem các ứng dụng khác nhau của nó. Ứng dụng đầu tiên cho phép chúng ta tìm các nút giống nhau nhất với 1 nút cho trước (theo độ tương đồng cosin):

```
print('Các nút giống nhau nhất với nút 0:')
để biết độ tương đồng trong model.wv.most_similar(positive = ['0']):
print(f' {similarity}')
```

Điều này tạo ra kết quả sau cho `Nút` giống nhau nhất với nút 0. 1 ứng dụng quan trọng khác là tính điểm tương đồng giữa 2 nút. Có thể thực hiện như sau:

```
# độ tương đồng giữa 2 nút
print(f"Độ tương đồng giữa nút 0 \& 4: {model.wv.similarity('0', '4')}")
```

Can plot resulting embeddings using *t-distributed stochastic neighbor embedding* (t-SNE) to visualize these high-dimensional vectors in 2D:

1. Import TSNE class from `sklearn`:

```
from sklearn.manifold import TSNE
```

2. Create 2 arrays: 1 to store word embeddings & the other 1 to store labels.
3. Next train t-SNE model with 2 dimension `n_components = 2` on embeddings.
4. Plot 2D vectors produced by trained t-SNE model with corresponding labels.

This plot is quite encouraging since we can see a clear line the separates 2 classes. It should be possible for a simple ML algorithm to classify these nodes with enough examples (training data). Implement a classifier & train it on our node embeddings:

1. Import a Random Forest model from `sklearn`, which is a popular choice when it comes to classification. Accuracy score is metric we will use to evaluate this model:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

2. Need to split embeddings into 2 groups: training & test data. A simple way of doing it: create masks as follows.
3. Next train Random Forest classifier on training data with appropriate labels.
4. Finally evaluate trained model on test data based on its accuracy score to give final result of our classifier.

– Có thể vẽ đồ thị các phép nhúng kết quả bằng cách sử dụng `nhúng lân cận ngẫu nhiên phân phối t` (t-SNE) để trực quan hóa các vectơ nhiều chiều này trong 2D:

1. Nhập lớp TSNE từ `sklearn`:

```
from sklearn.manifold import TSNE
```

2. Tạo 2 mảng: 1 để lưu trữ các phép nhúng từ & 1 để lưu trữ các nhãn.
3. Huấn luyện mô hình t-SNE tiếp theo với `n_components = 2` 2 chiều trên các phép nhúng.
4. Vẽ đồ thị các vectơ 2D được tạo ra bởi mô hình t-SNE đã huấn luyện với các nhãn tương ứng.

Đồ thị này khá đáng khích lệ vì chúng ta có thể thấy 1 đường rõ ràng phân tách 2 lớp. 1 thuật toán ML đơn giản có thể phân loại các nút này với đủ ví dụ (dữ liệu huấn luyện). Triển khai bộ phân loại & huấn luyện nó trên các nhúng nút của chúng ta:

1. Nhập 1 mô hình Rừng Ngẫu nhiên từ `sklearn`, đây là 1 lựa chọn phổ biến khi phân loại. Điểm chính xác là thước đo chúng ta sẽ sử dụng để đánh giá mô hình này:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

2. Cần chia các nhúng thành 2 nhóm: dữ liệu huấn luyện & dữ liệu kiểm tra. 1 cách đơn giản để thực hiện: tạo mặt nạ như sau.
3. Tiếp theo, huấn luyện bộ phân loại Rừng Ngẫu nhiên trên dữ liệu huấn luyện với các nhãn phù hợp.
4. Cuối cùng, đánh giá mô hình đã huấn luyện trên dữ liệu kiểm tra dựa trên điểm chính xác của nó để đưa ra kết quả cuối cùng cho bộ phân loại của chúng ta.

Our model obtains an accuracy score of 95.45%, which is pretty good considering unfavorable train/test split we gave it. There is still room for improvement, but this example showed 2 useful applications of DeepWalk:

* *Discovering similarities between nodes* using embeddings & cosine similarity (unsupervised learning)
* *Using these embeddings as a dataset* for a supervised task e.g. node classification.

See in following chaps, ability to learn node representations offers a lot of flexibility to design deeper & more complex architectures.

– Mô hình của chúng tôi đạt điểm chính xác là 95,45%, khá tốt khi xét đến việc phân tách bài kiểm tra train/bất lợi mà chúng tôi đã đưa ra. Vẫn còn nhiều điểm cần cải thiện, nhưng ví dụ này đã cho thấy 2 ứng dụng hữu ích của DeepWalk:

* *Khám phá điểm tương đồng giữa các nút* bằng cách sử dụng nhúng & độ tương đồng cosine (học không giám sát)
* *Sử dụng các nhúng này làm tập dữ liệu* cho 1 tác vụ có giám sát, e.g.: phân loại nút.

Xem trong các chương tiếp theo, khả năng học biểu diễn nút mang lại rất nhiều tính linh hoạt để thiết kế các kiến trúc sâu hơn & phức tạp hơn.

- Summary. In this chap, learned about DeepWalk architecture & its major components. Then transformed graph data into sequences using random walks to apply powerful Word2Vec algorithm. Resulting embeddings can be used to find similarities between nodes or as input to other algorithms. In particular, solved a node classification problem using a supervised approach.

  – Trong chương này, chúng ta đã tìm hiểu về kiến trúc DeepWalk & các thành phần chính của nó. Sau đó, dữ liệu đồ thị được chuyển đổi thành chuỗi bằng cách sử dụng các bước ngẫu nhiên để áp dụng thuật toán Word2Vec mạnh mẽ. Kết quả nhúng có thể được sử dụng để tìm điểm tương đồng giữa các nút hoặc làm đầu vào cho các thuật toán khác. Cụ thể, chúng ta đã giải quyết được bài toán phân loại nút bằng phương pháp tiếp cận có giám sát.

  In Chap. 4: Improving Embeddings with Biased Random Walks in Node2Vec, introduce a 2nd algorithm based on Word2Vec. Difference with DeepWalk: random walks can be biased towards more or less exploration, which directly impacts embeddings that are produced. Implement this algorithm on a new example & compare its representations with those obtained using DeepWalk.

  – Trong Chương 4: Cải thiện nhúng với bước ngẫu nhiên có thiên vị trong Node2Vec, giới thiệu thuật toán thứ 2 dựa trên Word2Vec. Điểm khác biệt với DeepWalk: bước ngẫu nhiên có thể thiên vị theo hướng khám phá nhiều hơn hoặc ít hơn, điều này ảnh hưởng trực tiếp đến các nhúng được tạo ra. Triển khai thuật toán này trên 1 ví dụ mới & so sánh các biểu diễn của nó với các biểu diễn thu được bằng DeepWalk.

PART 2: FUNDAMENTALS. In this 2nd part of book, delve into process of constructing node representations using graph learning. Start by exploring traditional graph learning techniques, drawing on advancements made in NLP. Our aim: understand how these techniques can be applied to graphs & how they can be used to build node representations.

– Trong phần 2 của cuốn sách này, chúng ta sẽ đi sâu vào quá trình xây dựng biểu diễn nút bằng phương pháp học đồ thị. Bắt đầu bằng việc khám phá các kỹ thuật học đồ thị truyền thống, dựa trên những tiến bộ trong NLP. Mục tiêu của chúng ta: tìm hiểu cách áp dụng các kỹ thuật này vào đồ thị & cách sử dụng chúng để xây dựng biểu diễn nút.

Then move on to incorporating node features into our models & explore how they can be used to build even more accurate representations. Finally, introduce 2 of most fundamental GNN architectures, Graph Convolutional Network (GCN) & Graph Attention Network (GAT). These 2 architectures are building blocks of many state-of-art graph learning methods & will provide a solid foundation for next part.

– Sau đó, chuyển sang việc tích hợp các đặc trưng nút vào mô hình của chúng ta & khám phá cách sử dụng chúng để xây dựng các biểu diễn chính xác hơn nữa. Cuối cùng, hãy giới thiệu 2 kiến trúc GNN cơ bản nhất: Mạng Tích chập Đồ thị (GCN) & Mạng Chú ý Đồ thị (GAT). Hai kiến trúc này là nền tảng của nhiều phương pháp học đồ thị tiên tiến & sẽ cung cấp nền tảng vững chắc cho phần tiếp theo.

By end of this part, have a deeper understanding of how traditional graph learning techniques, e.g. random walks, can be used to crate node representations & develops graph applications. Additionally, learn how to build even more powerful representations using GNNs. Introduce 2 key GNN architectures & learn how they can be used to tackle various graph-based tasks.

– Đến cuối phần này, bạn sẽ hiểu sâu hơn về cách sử dụng các kỹ thuật học đồ thị truyền thống, e.g. bước đi ngẫu nhiên, để tạo ra các biểu diễn nút & phát triển các ứng dụng đồ thị. Ngoài ra, hãy tìm hiểu cách xây dựng các biểu diễn mạnh mẽ hơn bằng GNN. Giới thiệu 2 kiến trúc GNN chính & tìm hiểu cách sử dụng chúng để giải quyết các tác vụ dựa trên đồ thị khác nhau.

- 4. Improving Embeddings with Biased Random Walks in Node2Vec. Node2Vec is an architecture largely based on DeepWalk. In prev chap,, saw 2 main components of this architecture: random walks & Word2Vec. How can improve quality of our embeddings? Interestingly enough, not with more ML. Instead, Node2Vec brings critical modifications to way random walks themselves are generated.

  – Cải thiện nhúng với bước ngẫu nhiên có thiên vị trong Node2Vec. Node2Vec là 1 kiến trúc phần lớn dựa trên DeepWalk. Trong chương trước, chúng ta đã thấy 2 thành phần chính của kiến trúc này: bước ngẫu nhiên & Word2Vec. Làm thế nào để cải thiện chất lượng nhúng của chúng ta? Điều thú vị là, không phải bằng cách tăng cường ML. Thay vào đó, Node2Vec mang đến những thay đổi quan trọng đối với cách tạo ra các bước ngẫu nhiên.

  In this chap, talk about these modifications & how to find best parameters for a given graph. Implement Node2Vec architecture & compare it to using DeepWalk on Zachary's Karate Club. This give you a good understanding of differences between 2 architectures. Finally, use this technology to build a real application: a move *recommender system (RecSys)* powered by Node2Vec.

  – Trong chương này, chúng ta sẽ thảo luận về những thay đổi này & cách tìm tham số tốt nhất cho 1 đồ thị nhất định. Triển khai kiến trúc Node2Vec & so sánh với việc sử dụng DeepWalk trên Zachary's Karate Club. Điều này giúp bạn hiểu rõ sự khác biệt giữa 2 kiến trúc. Cuối cùng, hãy sử dụng công nghệ này để xây dựng 1 ứng dụng thực tế: 1 hệ thống đề xuất di chuyển (RecSys) được hỗ trợ bởi Node2Vec.

  By end of this chap, know how to implement Node2Vec on any graph dataset & how to select good parameters. Understand why this architecture works better than DeepWalk in general, & how to apply it to build creative applications.

– Đến cuối chương này, bạn sẽ biết cách triển khai Node2Vec trên bất kỳ tập dữ liệu đồ thị nào & cách chọn tham số phù hợp. Hiểu lý do tại sao kiến trúc này hoạt động tốt hơn DeepWalk nói chung, & cách áp dụng nó để xây dựng các ứng dụng sáng tạo.

- Introducing Node2Vec. `Node2Vec` was introduced in 2016 by Grover & Leskovec from Stanford University [1]. It keeps same 2 main components from DeepWalk: random walks & Word2Vec. Difference: instead of obtaining sequences of nodes with a uniform distribution, random walks are carefully biased in Node2Vec. See why these *biased random walks* perform better & how to implement them in 2 following sects: defining a *neighborhood*, introducing biases in random walks. Start by questioning our intuitive concept of neighborhoods.

  – Giới thiệu Node2Vec. `Node2Vec` được giới thiệu vào năm 2016 bởi Grover & Leskovec từ Đại học Stanford [1]. Nó giữ nguyên 2 thành phần chính từ DeepWalk: các bước ngẫu nhiên & Word2Vec. Điểm khác biệt: thay vì thu được chuỗi các nút có phân phối đều, các bước ngẫu nhiên được phân bổ cẩn thận trong Node2Vec. Xem lý do tại sao các bước ngẫu nhiên *biased random walks* này hoạt động tốt hơn & cách triển khai chúng trong 2 phần sau: định nghĩa 1 *neighborhood*, giới thiệu các bước ngẫu nhiên trong các bước ngẫu nhiên. Bắt đầu bằng cách đặt câu hỏi về khái niệm trực quan của chúng ta về các vùng lân cận.

  * Defining a neighborhood. How do you define neighborhood of a node? Key concept introduced in Node2Vec is flexible notion of a neighborhood. Intuitively, we think of it as something close to initial node, but what does "close" mean in context of a graph? Want to explore 3 nodes in neighborhood of node A. This exploration process is also called a *sampling strategy*:
    · A possible solution would be to consider 3 closest nodes in terms of connections. In this case, neighborhood of A, noted $N(A)$, would be $\{B, C, D\}$.
    · Another possible sampling strategy consists of selecting nodes that are not adjacent to previous nodes 1st. In our example, $N(A) = \{D, E, F\}$.

    I.e., want to implement a BFS in 1st case & a DFS in 2nd one.
    – Định nghĩa vùng lân cận. Bạn định nghĩa vùng lân cận của 1 nút như thế nào? Khái niệm chính được giới thiệu trong Node2Vec là khái niệm linh hoạt về vùng lân cận. Theo trực giác, chúng ta nghĩ về nó như 1 thứ gì đó gần với nút ban đầu, nhưng "đóng" có nghĩa là gì trong ngữ cảnh của đồ thị? Bạn muốn khám phá 3 nút trong vùng lân cận của nút A. Quá trình khám phá này cũng được gọi là *chiến lược lấy mẫu*:
    · 1 giải pháp khả thi là xem xét 3 nút gần nhất về mặt kết nối. Trong trường hợp này, vùng lân cận của A, ký hiệu là $N(A)$, sẽ là $\{B, C, D\}$.
    · 1 chiến lược lấy mẫu khả thi khác bao gồm việc chọn các nút không liền kề với các nút trước đó trước. Trong ví dụ của chúng ta, $N(A) = \{D, E, F\}$.

    E.g., bạn muốn triển khai BFS trong trường hợp thứ nhất & DFS trong trường hợp thứ 2.
    What is important to notice here: these sampling strategies have opposite behaviors: BFS focuses on local network around a node while DFS establishes a more macro view of graph. Considering our intuitive definition of a neighborhood, it is temping to simply discard DFS. However, Node2vec's authors argue: this would be a mistake: each approach captures a different but valuable representation of network.
    – Điều quan trọng cần lưu ý ở đây: các chiến lược lấy mẫu này có hành vi trái ngược nhau: BFS tập trung vào mạng cục bộ xung quanh 1 nút trong khi DFS thiết lập 1 góc nhìn vĩ mô hơn về đồ thị. Xét theo định nghĩa trực quan của chúng ta về vùng lân cận, việc loại bỏ DFS là điều dễ hiểu. Tuy nhiên, các tác giả của Node2vec lập luận: đây sẽ là 1 sai lầm: mỗi phương pháp nắm bắt 1 biểu diễn mạng khác nhau nhưng có giá trị.
    They argue: BFS is ideal to emphasize structural equivalence since this strategy only looks at neighboring nodes. In these random walks, nodes are often repeated & stay close to each other. DFS, on other hand, emphasizes opposite of homophily by creating sequences of distant nodes. These random walks can sample nodes that are far from source & thus become less representative. This is why we are looking for a trade-off between these 2 properties: homophily may be more helpful for understanding certain graphs & vice versa.
    – Họ lập luận: BFS là lý tưởng để nhấn mạnh tính tương đương về mặt cấu trúc vì chiến lược này chỉ xem xét các nút lân cận. Trong các bước đi ngẫu nhiên này, các nút thường được lặp lại & ở gần nhau. Mặt khác, DFS nhấn mạnh tính đối lập của tính đồng dạng bằng cách tạo ra các chuỗi nút ở xa. Các bước đi ngẫu nhiên này có thể lấy mẫu các nút ở xa nguồn & do đó trở nên kém đại diện hơn. Đây là lý do tại sao chúng ta đang tìm kiếm sự đánh đổi giữa 2 tính chất này: tính đồng dạng có thể hữu ích hơn trong việc hiểu 1 số đồ thị & ngược lại.
    If you are confused about this connection, you are not alone: several papers & blogs wrongly assume: BFS emphasizes homophily & DFS is connected to structural equivalence. In any case, consider graphs that combine homophily & structural equivalence to be desired solution. This is why, regardless of these connections, want to use both sampling strategies to create our dataset. See how can implement them to generate random walks.
    – Nếu bạn đang bối rối về mối liên hệ này, bạn không phải là người duy nhất: 1 số bài báo & blog đã sai lầm khi cho rằng: BFS nhấn mạnh tính đồng dạng & DFS có liên quan đến tính tương đương về mặt cấu trúc. Trong mọi trường hợp, hãy coi các đồ thị kết hợp tính đồng dạng & tính tương đương về mặt cấu trúc là giải pháp mong muốn. Đây là lý do tại sao, bất kể những mối liên hệ này, chúng ta nên sử dụng cả 2 chiến lược lấy mẫu để tạo tập dữ liệu. Hãy xem cách triển khai chúng để tạo các bước ngẫu nhiên.

  * Introducing biases in random walks. As a reminder, random walks are sequences of nodes that are randomly selected in a graph. They have a starting point, which can also be random, & a predefined length. Nodes that often appear together in

these walks are like words that appear together in sentences: under homophily hypothesis, they share a similar meaning, hence a similar representation.
– Giới thiệu về độ lệch trong các bước ngẫu nhiên. Xin nhắc lại, các bước ngẫu nhiên là chuỗi các nút được chọn ngẫu nhiên trong đồ thị. Chúng có điểm khởi đầu, cũng có thể là ngẫu nhiên, & độ dài được xác định trước. Các nút thường xuất hiện cùng nhau trong các bước này giống như các từ xuất hiện cùng nhau trong câu: theo giả thuyết đồng dạng, chúng có ý nghĩa tương tự nhau, do đó có biểu diễn tương tự nhau.
In Node2Vec, goal: bias randomness of these walks to either 1 of following:

1. Promoting nodes that are not connected to previous one (similar to DFS)
2. Promoting nodes that are close to prevous one (similar to BFS)/

Current node is called j, previous node is i, & future node is k. Note π_{jk} unnormalized transition probability from node j to node k. This probability can be decomposed as $\pi_{jk} = \alpha(i,k) \cdot \omega_{jk}$, where $\alpha(i,k)$ is *search bias* between nodes i, k & ω_{jk} is weight of edge from j to k.
– Trong Node2Vec, mục tiêu: điều chỉnh độ ngẫu nhiên của các bước này thành 1 trong các bước sau:

1. Thăng hạng các nút không được kết nối với nút trước đó (tương tự như DFS)
2. Thăng hạng các nút gần nút trước đó (tương tự như BFS)/

Nút hiện tại được gọi là j, nút trước đó là i, & nút tương lai là k. Lưu ý π_{jk} xác suất chuyển tiếp không chuẩn hóa từ nút j đến nút k. Xác suất này có thể được phân tích thành $\pi_{jk} = \alpha(i,k) \cdot \omega_{jk}$, trong đó $\alpha(i,k)$ là *độ lệch tìm kiếm* giữa các nút i, k & ω_{jk} là trọng số của cạnh từ j đến k.
In DeepWalk, have $\alpha(a,b) = 1$ for any pair of nodes a, b. In Node2Vec, value of $\alpha(a,b)$ is defined based on distance between nodes & 2 additional parameters: p: return parameter & q: in-out parameter. Their role: approximate DFS & BFS, resp. Here is how value of $\alpha(a,b)$ is defined:

$$\alpha(a,b) = \begin{cases} \frac{1}{p} & \text{if } d_{ab} = 0, \\ 1 & \text{if } d_{ab} = 1, \\ \frac{1}{q} & \text{if } d_{ab} = 2. \end{cases}$$

Here d_{ab} is shortest path distance between nodes a, b. Can update unnormalized transition probability from previous graphs as follows: Fig. 4.3: Graph with transition probabilities. Decrypt these probabilities:

· Walk starts from node i & now arrive at node j. Probability of going back to previous node i is controlled by parameter p. Higher it is, more random walk will explore new nodes instead of repeating same ones & looking like DFS.
· Unnormalized probability of going to k_1 is 1 because this node is in immediate neighborhood of our previous node i.
· Finally, probability of going to node k_2 is controlled by parameter q. Higher it is, more random walk will focus on nodes that are close to previous one & look like BFS.
Best way to understand this is to actually implement this architecture & play with parameters. Do it step by step on Zachary's Karate Club.

– Ở đây d_{ab} là khoảng cách đường đi ngắn nhất giữa các nút a, b. Có thể cập nhật xác suất chuyển tiếp chưa chuẩn hóa từ các đồ thị trước đó như sau: Hình 4.3: Đồ thị với xác suất chuyển tiếp. Giải mã các xác suất này:

· Đường đi bắt đầu từ nút i & bây giờ đến nút j. Xác suất quay lại nút i trước đó được kiểm soát bởi tham số p. Giá trị p càng cao, bước đi ngẫu nhiên càng khám phá các nút mới thay vì lặp lại các nút cũ & trông giống như DFS.
· Xác suất chưa chuẩn hóa để đi đến k_1 là 1 vì nút này nằm trong vùng lân cận trực tiếp của nút i trước đó.
· Cuối cùng, xác suất đi đến nút k_2 được kiểm soát bởi tham số q. Giá trị p càng cao, bước đi ngẫu nhiên càng tập trung vào các nút gần nút trước đó & trông giống như BFS.
Cách tốt nhất để hiểu điều này là thực sự triển khai kiến trúc này & thử nghiệm với các tham số. Thực hiện từng bước trên Câu lạc bộ Karate của Zachary.

Note: it is an unweighted network, which is why transition probability is only determined by search bias. 1st, want to create a function that will randomly select next node in a graph based on previous node, current node, & 2 parameters p, q.

1. Start by importing required libraries: `networkx, random, numpy`.
2. Define `next_node` function with list of our parameters.
3. Retrieve list of neighboring nodes from current node & initialize a list of alpha values.
4. For each neighbor, want to calculate appropriate alpha value: $\frac{1}{p}$ if this neighbor is previous node, 1 if this neighbor is connected to previous node, & $\frac{1}{q}$ otherwise.
5. Normalize these values to create probabilities.
6. Randomly select next node based on transition probabilities calculated in previous step using `np.random.choice()` & return it.

Before this function can be tested, need code to generate entire random walk.
– Lưu ý: đây là 1 mạng không trọng số, đó là lý do tại sao xác suất chuyển tiếp chỉ được xác định bởi độ lệch tìm kiếm. Đầu tiên, muốn tạo 1 hàm sẽ chọn ngẫu nhiên nút tiếp theo trong đồ thị dựa trên nút trước đó, nút hiện tại, & 2 tham số p, q.

1. Bắt đầu bằng cách nhập các thư viện cần thiết: `networkx, random, numpy`.
2. Định nghĩa hàm `next_node` với danh sách các tham số của chúng ta.
3. Truy xuất danh sách các nút lân cận từ nút hiện tại & khởi tạo danh sách các giá trị alpha.
4. Đối với mỗi nút lân cận, muốn tính giá trị alpha phù hợp: $\frac{1}{p}$ nếu nút lân cận này là nút trước đó, 1 nếu nút lân cận này được kết nối với nút trước đó, & $\frac{1}{q}$ nếu không.
5. Chuẩn hóa các giá trị này để tạo xác suất.
6. Chọn ngẫu nhiên nút tiếp theo dựa trên xác suất chuyển tiếp được tính toán ở bước trước bằng cách sử dụng `np.random.choice()` & trả về kết quả.

Trước khi có thể kiểm tra hàm này, cần mã để tạo toàn bộ bước ngẫu nhiên.
Way we generate these random walks is similar to what we saw in previous chap. Difference: next node is chosen by `next_node()` function, which requires additional parameters: p, q, but also previous & current nodes. These nodes can easily be obtained by looking at 2 last elements added to `walk` variable. Also return strings instead of integers for compatibility reasons. Here is new version of `random_walk()` function.
– Cách chúng ta tạo ra các bước ngẫu nhiên này tương tự như những gì chúng ta đã thấy trong chương trước. Điểm khác biệt: nút tiếp theo được chọn bởi hàm `next_node()`, hàm này yêu cầu các tham số bổ sung: p, q, nhưng cũng bao gồm các nút trước đó & nút hiện tại. Các nút này có thể dễ dàng được lấy bằng cách xem xét 2 phần tử cuối cùng được thêm vào biến `walk`. Đồng thời, trả về chuỗi thay vì số nguyên vì lý do tương thích. Dưới đây là phiên bản mới của hàm `random_walk()`.
Now have every element to generate our random walks. Try one with a length of 5, $p = q = 1$: `random_walk(0, 8, p = 1, q = 1)`. This should be random since every neighboring node has same transition probability. With these parameters, reproduce exact DeepWalk algorithm. Now bias them toward going back to previous node with $q = 10$: `random_walk(0, 8, p = 1, q = 10)`. This time, random walk explores more node in graph, can see that it never goes back to previous node because probability is low with $p = 10$: `random_walk(0, 8, p = 10, q = 1)`. See how to use these properties in a real example & compare it to DeepWalk.
– Bây giờ, hãy tạo ra mọi phần tử để tạo ra các bước đi ngẫu nhiên. Hãy thử 1 bước đi có độ dài 5, $p = q = 1$: `random_walk(0, 8, p = 1, q = 1)`. Bước đi này phải ngẫu nhiên vì mọi nút lân cận đều có cùng xác suất chuyển tiếp. Với các tham số này, hãy tái tạo chính xác thuật toán DeepWalk. Bây giờ, hãy hướng chúng về phía nút trước đó với $q = 10$: `random_walk(0, 8, p = 1, q = 10)`. Lần này, bước đi ngẫu nhiên khám phá nhiều nút hơn trong đồ thị, có thể thấy rằng nó không bao giờ quay lại nút trước đó vì xác suất thấp với $p = 10$: `random_walk(0, 8, p = 10, q = 1)`. Xem cách sử dụng các thuộc tính này trong 1 ví dụ thực tế & so sánh nó với DeepWalk.

- Implementing Node2Vec. Now have function to generate biased random walks, implementation of Node2Vec is very similar to implementing DeepWalk. It is so similar that we can reuse same code & create sequences with $p = q = 1$ to implement DeepWalk as a special case of Node2Vec. Reuse Zachary's Karate Club for this task. As in prev chap, goal: correctly classify each member of club as part of 1 of 2 groups ("Mr. Hi" & "Officer"). Use node embeddings provided by Node2Vec as input to a ML classifier (Random Forest in this case). See how to implement it step by step:

1. 1st, want to install `gensim` library to use Word2Vec. This time, use version 3.8.0 for compatibility reasons:
   ```
   !pip install -qI gensim==3.8.0
   ```
2. Import required libraries.
3. Load dataset (Zachary's Karate Club).
4. Transform nodes' labels into numerical values: 0 & 1.
5. Generate a list of random walks as seen previously using `random_walk()` function 80 times for each node in graph. Parameters p, q as specified here (2 & 1, resp).
6. Create an instance of Word2Vec (a skip-gram model) with a hierarchical `softmax` function.
7. Skip-gram model is trained on sequences generated for 30 epochs.
8. Create masks to train & test classifier.
9. Random Forest classifier is trained on training data.
10. Evaluate it in terms of accuracy for test data.

– Triển khai Node2Vec. Giờ đây, chúng ta đã có chức năng tạo các bước ngẫu nhiên có thiên vị, việc triển khai Node2Vec rất giống với việc triển khai DeepWalk. Nó giống nhau đến mức chúng ta có thể sử dụng lại cùng 1 mã & tạo các chuỗi với $p = q = 1$ để triển khai DeepWalk như 1 trường hợp đặc biệt của Node2Vec. Hãy sử dụng lại Câu lạc bộ Karate của Zachary cho nhiệm vụ này. Như trong chương trước, mục tiêu: phân loại chính xác từng thành viên của câu lạc bộ thành 1 trong 2 nhóm ("Mr. Hi" & "Sĩ quan"). Sử dụng các nhúng nút do Node2Vec cung cấp làm đầu vào cho bộ phân loại ML (trong trường hợp này là Random Forest). Xem cách triển khai từng bước:

1. Trước tiên, muốn cài đặt thư viện `gensim` để sử dụng Word2Vec. Lần này, hãy sử dụng phiên bản 3.8.0 vì lý do tương thích:
   ```
   !pip install -qI gensim==3.8.0
   ```
2. Nhập các thư viện cần thiết.

3. Tải tập dữ liệu (Câu lạc bộ Karate Zachary).
4. Chuyển đổi nhãn của các nút thành giá trị số: 0 & 1.
5. Tạo danh sách các bước đi ngẫu nhiên như đã thấy trước đó bằng cách sử dụng hàm `random_walk()` 80 lần cho mỗi nút trong đồ thị. Các tham số p, q như được chỉ định ở đây (tương ứng là 2 & 1).
6. Tạo 1 thể hiện của Word2Vec (mô hình skip-gram) với hàm `softmax` phân cấp.
7. Mô hình skip-gram được huấn luyện trên các chuỗi được tạo trong 30 kỷ nguyên.
8. Tạo mặt nạ để huấn luyện & kiểm tra bộ phân loại.
9. Bộ phân loại Rừng ngẫu nhiên được huấn luyện trên dữ liệu huấn luyện.
10. Đánh giá nó về độ chính xác cho dữ liệu kiểm tra.

To implement DeepWalk, can repeat exact same process with $p = q = 1$. However, to make a fair comparison, cannot use a single accuracy score. Indeed, there are a lot of stochastic processes involved – could be unlucky & get a better result from worst model.

– Để triển khai DeepWalk, có thể lặp lại chính xác quy trình đó với $p = q = 1$. Tuy nhiên, để so sánh 1 cách công bằng, không thể sử dụng 1 điểm chính xác duy nhất. Thực tế, có rất nhiều quy trình ngẫu nhiên liên quan – có thể không may mắn & có được kết quả tốt hơn từ mô hình tệ nhất.

To limit randomness of our results, can repeat this process 100 times & take mean value. This result is a lot more stable & can even include standard deviation (using `np.std()`) to measure variability in accuracy scores.

– Để hạn chế tính ngẫu nhiên của kết quả, chúng ta có thể lặp lại quy trình này 100 lần & lấy giá trị trung bình. Kết quả này ổn định hơn nhiều & thậm chí có thể bao gồm độ lệch chuẩn (sử dụng `np.std()`) để đo lường độ biến thiên của điểm chính xác.

But just before we do that, play a game. In prev chap, talked about Zachary's Karate Club as a homophilic network. This property is emphasized by DFS, which is encouraged by increasing parameter p. If this statement & connection between DFS & homophily are true, should get better results with higher values of p.

– Nhưng trước khi làm điều đó, hãy chơi 1 trò chơi. Trong chương trước, chúng ta đã thảo luận về Câu lạc bộ Karate của Zachary như 1 mạng lưới đồng dạng. Tính chất này được nhấn mạnh bởi DFS, & được hỗ trợ bằng cách tăng tham số p. Nếu phát biểu này & kết nối giữa DFS & đồng dạng là đúng, sẽ cho kết quả tốt hơn với các giá trị p cao hơn.

Repeated same experiment for values of $p, q \in [1, 7]$. In a real ML project, would use validation data to perform this parameter search. In this example, use test data because this study is already our final application. Fig. 4.5: Average accuracy score & standard deviation for different values of p, q. There are several noticeable results:

* DeepWalk $p = q = 14$ performs worse than any other combination of p, q that is covered here. This is true for this dataset & show how useful biased random walks can be. However, it is not always case: non-biased random walks can also perform better on other datasets.
* High values of p lead to better performance, which validates our hypothesis. Knowing that this is a social network strongly suggests that biasing our random walks toward homophily is a good strategy. This is something to keep in mind when dealing with this kind of graph.
  Feel free to play with parameters & try to find other interesting results. Could explore results with very high values of $p > 7$, or on contrary, values of p, q between 0, 1.

Zachary's Karate Club is a basic dataset, but see in next sect how can use this technology to build much more interesting applications.

– Lặp lại cùng 1 thí nghiệm với các giá trị $p, q \in [1, 7]$. Trong 1 dự án ML thực tế, sẽ sử dụng dữ liệu xác thực để thực hiện tìm kiếm tham số này. Trong ví dụ này, hãy sử dụng dữ liệu thử nghiệm vì nghiên cứu này đã là ứng dụng cuối cùng của chúng tôi. Hình 4.5: Điểm chính xác trung bình & độ lệch chuẩn cho các giá trị khác nhau của p, q. Có 1 số kết quả đáng chú ý:

* DeepWalk $p = q = 14$ hoạt động kém hơn bất kỳ tổ hợp p, q nào khác được đề cập ở đây. Điều này đúng với tập dữ liệu này & cho thấy các bước đi ngẫu nhiên có thiên vị hữu ích như thế nào. Tuy nhiên, không phải lúc nào cũng vậy: các bước đi ngẫu nhiên không thiên vị cũng có thể hoạt động tốt hơn trên các tập dữ liệu khác.
* Các giá trị p cao dẫn đến hiệu suất tốt hơn, điều này xác nhận giả thuyết của chúng tôi. Việc biết rằng đây là 1 mạng xã hội cho thấy rõ ràng rằng việc định hướng các bước đi ngẫu nhiên của chúng tôi theo hướng đồng dạng là 1 chiến lược tốt. Đây là điều cần lưu ý khi xử lý loại đồ thị này.
  Hãy thoải mái thử nghiệm với các tham số & cố gắng tìm ra những kết quả thú vị khác. Bạn có thể khám phá các kết quả với giá trị rất cao $p > 7$, hoặc ngược lại, các giá trị p, q nằm trong khoảng từ 0 đến 1.

Câu lạc bộ Karate của Zachary là 1 tập dữ liệu cơ bản, nhưng hãy xem trong phần tiếp theo cách sử dụng công nghệ này để xây dựng các ứng dụng thú vị hơn nhiều.

- Building a move RecSys. 1 of most popular applications of GNNs is RecSys. If think about foundation of Word2Vec (&, thus, DeepWalk & Node2Vec), goal: produce vectors with ability to measure their similarity. Encode movies instead of words, & can suddenly ask for movies that are most similar to a given input file. It sounds a lot like a RecSys.

  – Xây dựng RecSys di chuyển. 1 trong những ứng dụng phổ biến nhất của GNN là RecSys. Nếu nghĩ về nền tảng của Word2Vec (&, i.e., DeepWalk & Node2Vec), mục tiêu: tạo ra các vectơ có khả năng đo lường độ tương đồng của chúng. Mã hóa phim thay vì từ ngữ, & có thể đột nhiên yêu cầu các phim giống nhất với 1 tệp đầu vào nhất định. Nghe có vẻ rất giống RecSys.

But how to encode movies? Want to create (biased) random walks of movies, but this requires a graph dataset where similar movies are connected to each other. This is not easy to find.

– Nhưng làm thế nào để mã hóa phim? Bạn muốn tạo các chuỗi phim ngẫu nhiên (có thiên vị), nhưng điều này đòi hỏi 1 tập dữ liệu đồ thị trong đó các phim tương tự được kết nối với nhau. Điều này không dễ tìm.

Another approach is to look at user ratings. There are different techniques to build a graph based on ratings: bipartite graphs, edges based on pointwise mutual information, & so on. In this sect, implement a simple & intuitive approach: movies that are likely by same users are connected. Then use this graph to learn movie embeddings using Node2Vec:

pp. 60–64+++

- Summary. In this chap, learned about Node2Vec, a 2nd architecture based on popular Word2Vec. Implemented functions to generate biased random walks & explained connection between their parameters & 2 network properties: homophily & structural equivalence. Showed their usefulness by comparing Node2Vec's results to DeepWalk's for Zachary's Karate Club. Finally, build our 1st RecSys using a custom graph dataset & another implementation of Node2Vec. It gave us correct recommendations that we will improve even more in later chaps.

  – Trong chương này, chúng ta đã tìm hiểu về Node2Vec, 1 kiến trúc thứ 2 dựa trên Word2Vec phổ biến. Chúng tôi đã triển khai các hàm để tạo các bước ngẫu nhiên có thiên vị & giải thích mối liên hệ giữa các tham số của chúng & 2 thuộc tính mạng: đồng dạng & tương đương cấu trúc. Chúng tôi đã chứng minh tính hữu ích của chúng bằng cách so sánh kết quả của Node2Vec với kết quả của DeepWalk cho Câu lạc bộ Karate của Zachary. Cuối cùng, chúng tôi đã xây dựng RecSys đầu tiên của mình bằng cách sử dụng 1 bộ dữ liệu đồ thị tùy chỉnh & 1 triển khai khác của Node2Vec. Nó đã đưa ra cho chúng tôi những khuyến nghị chính xác mà chúng tôi sẽ cải thiện hơn nữa trong các chương sau.

  In Chap. 5: Including Node Features with Vanilla Neural Networks, talk about 1 overlooked issue concerning DeepWalk & Node2Vec: lack of proper node features. Try to address this problem using traditional neural networks, which cannot understand network topology. This dilemma is important to understand before we finally introduce answer: graph neural networks.

  – Trong Chương 5: Bao gồm các đặc điểm nút với mạng nơ-ron Vanilla, chúng ta sẽ thảo luận về 1 vấn đề thường bị bỏ qua liên quan đến DeepWalk & Node2Vec: thiếu các đặc điểm nút phù hợp. Hãy thử giải quyết vấn đề này bằng cách sử dụng các mạng nơ-ron truyền thống, vốn không thể hiểu được cấu trúc mạng. Vấn đề nan giải này rất quan trọng cần được hiểu rõ trước khi chúng ta giới thiệu câu trả lời cuối cùng: mạng nơ-ron đồ thị.

- 5. Including Node Features with Vailla Neural Networks. So far, only type of information we have considered is graph topology. However, graph datasets tend to be richer than a mere set of connections: nodes & edges can also have features to represent scores, colors, words, & so on. Including this additional information in our input data is essential to produce best embeddings possible. In fact, this is something natural in ML: node & edge featues have same structure as a tabular (non-graph) dataset. I.e., traditional techniques can be applied to this data, e.g. neural networks.

  – Cho đến nay, loại thông tin duy nhất chúng ta xem xét là tôpô đồ thị. Tuy nhiên, tập dữ liệu đồ thị thường phong phú hơn 1 tập hợp các kết nối đơn thuần: các nút & cạnh cũng có thể có các đặc trưng để biểu diễn điểm số, màu sắc, từ ngữ, v.v. Việc đưa thông tin bổ sung này vào dữ liệu đầu vào là điều cần thiết để tạo ra các phép nhúng tốt nhất có thể. Trên thực tế, đây là điều tự nhiên trong ML: các đặc trưng nút & cạnh có cùng cấu trúc với tập dữ liệu dạng bảng (không phải đồ thị). Tức là, các kỹ thuật truyền thống có thể được áp dụng cho dữ liệu này, e.g., như mạng nơ-ron.

  In this chap, introduce 2 new graph datasets: `Cora & Facebook Page-Page`. See how Vanilla Neural Networks perform on node features only by considering them as tabular datasets. Then experiment to include topological information in our neural networks. This will give us 1st GNN architecture: a simple model that considers both node features & edges. Finally, compare performance of 2 architectures & obtain 1 of most important results of this book.

  – Trong chương này, chúng ta sẽ giới thiệu 2 bộ dữ liệu đồ thị mới: `Cora & Facebook Page-Page`. Xem cách Mạng Nơ-ron Vanilla hoạt động trên các đặc điểm nút chỉ bằng cách xem xét chúng dưới dạng tập dữ liệu dạng bảng. Sau đó, hãy thử nghiệm việc đưa thông tin tôpô vào mạng nơ-ron của chúng ta. Điều này sẽ cho chúng ta kiến trúc GNN đầu tiên: 1 mô hình đơn giản xem xét cả đặc điểm nút & cạnh. Cuối cùng, hãy so sánh hiệu suất của 2 kiến trúc & thu được 1 trong những kết quả quan trọng nhất của cuốn sách này.

  By end of this chap, master implementation of vanilla neural networks & vanilla GNNs in PyTorch. Able to embed topological features into node representations, which is basis of every GNN architecture. This will allow you to greatly improve performance of your models by transforming tabular datasets into graphs problems.

  – Đến cuối chương này, bạn sẽ thành thạo việc triển khai mạng nơ-ron nhân tạo thuần túy & mạng nơ-ron nhân tạo thuần túy (GNN) trong PyTorch. Bạn có thể nhúng các đặc điểm tôpô vào biểu diễn nút, vốn là nền tảng của mọi kiến trúc GNN. Điều này sẽ cho phép bạn cải thiện đáng kể hiệu suất mô hình bằng cách chuyển đổi các tập dữ liệu dạng bảng thành các bài toán đồ thị.

  - Introducing graph datasets. Graph dataset used in this chap are richer than Zachary's Karate Club: they have more nodes, more edges, & include node features. In this sect, introduce them to give us a good understanding of these graphs & how to process them with PyTorch Geometric. Here are 2 datasets we will use: `Cora` dataset, `Facebook Page-page` dataset. Start with smaller one: popular `Cora` dataset.

    – Giới thiệu về bộ dữ liệu đồ thị. Bộ dữ liệu đồ thị được sử dụng trong chương này phong phú hơn Zachary's Karate Club: chúng có nhiều nút hơn, nhiều cạnh hơn, & bao gồm các đặc trưng nút. Trong phần này, chúng tôi sẽ giới thiệu chúng để

giúp chúng ta hiểu rõ hơn về các đồ thị này & cách xử lý chúng bằng PyTorch Geometric. Dưới đây là 2 bộ dữ liệu chúng ta sẽ sử dụng: bộ dữ liệu `Cora`, bộ dữ liệu `Facebook Page-page`. Bắt đầu với bộ dữ liệu nhỏ hơn: bộ dữ liệu `Cora` phổ biến.

* Cora dataset. Introduced by Sen et al. in 2008 [1], `Cora` (no license) is most popular dataset for node classification in scientific literature. It represents a network of 2708 publications, where each connection is a reference. Each publication is described as a binary vector of 1433 unique words, where 0 & 1 indicate absence or presence of corresponding word, resp. This representation is also called a binary *bag of words* in NLP. Goal: classify each node into 1 of 7 categories.
  – `Cora dataset.` Được giới thiệu bởi Sen & cộng sự vào năm 2008 [1], `Cora` (không có giấy phép) là tập dữ liệu phổ biến nhất để phân loại nút trong các tài liệu khoa học. Nó đại diện cho 1 mạng lưới gồm 2708 ấn phẩm, trong đó mỗi kết nối là 1 tham chiếu. Mỗi ấn phẩm được mô tả dưới dạng 1 vectơ nhị phân gồm 1433 từ duy nhất, trong đó 0 & 1 biểu thị sự có mặt hoặc vắng mặt của từ tương ứng. Biểu diễn này còn được gọi là `túi từ` nhị phân trong NLP. Mục tiêu: phân loại mỗi nút thành 1 trong 7 loại.
  Regardless of type of data, visualization is always an important step to getting a good grasp of problem we face. However, graphs can quickly become too big to visualize using Python libraries e.g. `networkx`. This is why dedicated tools have been developed specifically for graph data visualization. In this book, utilize 2 of most popular ones: yEd Live (https://www.yworks.com/yed-live/) & Gephi (https://gephi.org/).
  – Bất kể loại dữ liệu nào, trực quan hóa luôn là 1 bước quan trọng để nắm bắt tốt vấn đề chúng ta đang gặp phải. Tuy nhiên, đồ thị có thể nhanh chóng trở nên quá lớn để trực quan hóa bằng các thư viện Python, ví dụ: `networkx`. Đây là lý do tại sao các công cụ chuyên dụng đã được phát triển dành riêng cho việc trực quan hóa dữ liệu đồ thị. Trong cuốn sách này, chúng tôi sẽ sử dụng 2 công cụ phổ biến nhất: yEd Live (https://www.yworks.com/yed-live/) & Gephi (https://gephi.org/).
  Following Fig. 5.1: Cora dataset visualized with yEd Live. is a plot of `Cora` dataset made with yEd Live. Can see nodes corresponding to papers in orange & connections between them in green. Some papers are so interconnected that they form clusters. These clusters should be easier to classify than poorly connected nodes.
  – Hình dưới đây Hình 5.1: Tập dữ liệu Cora được trực quan hóa bằng yEd Live. là sơ đồ của tập dữ liệu `Cora` được tạo bằng yEd Live. Có thể thấy các nút tương ứng với các bài báo màu cam & các kết nối giữa chúng màu xanh lá cây. 1 số bài báo được kết nối chặt chẽ đến mức chúng tạo thành các cụm. Các cụm này sẽ dễ phân loại hơn so với các nút có kết nối kém.
  Import it & analyze its main characteristics with PyTorch Geometric. This library has a dedicated class to download dataset & return a relevant data structure. Assume here: PyTorch Geometric has already been installed:
  1. Import `Planetoid` class from PyTorch Geometric.
  2. Cora only has one graph we can store in a dedicated data variable `data = dataset[0]`
  3. Print information about dataset in general & can also get detailed information thanks to dedicated functions from PyTorch Geometric.

  1st output confirms information about number of nodes, features, & classes. 2nd one gives more insights into graph itself: edges are undirected, every node has neighbors, & graph does not have any self-loop. Could test other properties using PyTorch Geometric's utils functions, but we would not learn anything new in this example.
  – Nhập & phân tích các đặc điểm chính của nó bằng PyTorch Geometric. Thư viện này có 1 lớp chuyên dụng để tải xuống tập dữ liệu & trả về 1 cấu trúc dữ liệu phù hợp. Giả sử ở đây: PyTorch Geometric đã được cài đặt:
  1. Nhập lớp `Planetoid` từ PyTorch Geometric.
  2. Cora chỉ có 1 đồ thị mà chúng ta có thể lưu trữ trong 1 biến dữ liệu chuyên dụng `data = dataset[0]`
  3. In thông tin chung về tập dữ liệu & cũng có thể lấy thông tin chi tiết nhờ các hàm chuyên dụng từ PyTorch Geometric.

  Đầu ra đầu tiên xác nhận thông tin về số lượng nút, đặc điểm, & lớp. Đầu ra thứ hai cung cấp thêm thông tin chi tiết về bản thân đồ thị: các cạnh không có hướng, mỗi nút đều có hàng xóm, & đồ thị không có vòng lặp tự thân. Có thể kiểm tra các thuộc tính khác bằng cách sử dụng các hàm tiện ích của PyTorch Geometric, nhưng chúng ta sẽ không học được điều gì mới trong ví dụ này.
  Now know more about `Cora`, see one that is more representative of size of real-world social networks: Facebook `Page-Page` dataset.
  – Bây giờ hãy tìm hiểu thêm về `Cora`, hãy xem 1 ví dụ đại diện hơn cho quy mô của mạng xã hội trong thế giới thực: Tập dữ liệu `Page-Page` của Facebook.
* Facebook Page-Page dataset. This dataset was introduced by Rozemberczki et al. in 2019 [2]. It was created using Facebook Graph API in Nov 2017. In this dataset, each of 22,470 nodes represents an official Facebook page. Pages are connected when there are mutual likes between them. Node features (128-dim vectors) are created from textual descriptions written by the owners of these pages. Goal: classify each node into 1 of 4 categories: politicians, companies, television shows, & governmental organizations.
  – Facebook Page-Page dataset. Bộ dữ liệu này được Rozemberczki & cộng sự giới thiệu vào năm 2019 [2]. Nó được tạo bằng Facebook Graph API vào tháng 11 năm 2017. Trong bộ dữ liệu này, mỗi nút trong số 22.470 nút đại diện cho 1 trang Facebook chính thức. Các trang được kết nối khi có lượt thích lẫn nhau giữa chúng. Các đặc trưng nút (vectơ 128-dim) được tạo từ các mô tả văn bản do chủ sở hữu của các trang này viết. Mục tiêu: phân loại mỗi nút thành 1 trong 4 loại: chính trị gia, công ty, chương trình truyền hình, & tổ chức chính phủ.
  Facebook `Page-Page` dataset is similar to previous one: it is a social network with a node classification task. However, there are 3 major differences with `Cora`:
  · Number of nodes is much higher (2,708 versus 22,470).

· Dimensionality of node features decreased dramatically (from 1,433 to 128)

· Goal: classify each node into 4 categories instead of 7 (which is easier since there are fewer options)

Following Fig. 5.2: Facebook Page-Page dataset visualized with Gephi is a visualization of dataset using Gephi. 1st, nodes with few connections have been filtered out to improve performance. Size of remaining nodes depends on their number of connections, & their color indicates category they belong to. Finally, 2 layouts have been applied: Fruchterman-Reingold & ForceAtlas2.

– Bộ dữ liệu Facebook `Page-Page` tương tự như bộ dữ liệu trước: đây là 1 mạng xã hội với nhiệm vụ phân loại nút. Tuy nhiên, có 3 điểm khác biệt chính với `Cora`:

· Số lượng nút cao hơn nhiều (2.708 so với 22.470).

· Số chiều của các đặc trưng nút giảm đáng kể (từ 1.433 xuống 128)

· Mục tiêu: phân loại mỗi nút thành 4 loại thay vì 7 loại (dễ hơn vì có ít tùy chọn hơn)

Sau đây là Hình 5.2: Bộ dữ liệu Facebook Page-Page được trực quan hóa bằng Gephi là 1 hình ảnh trực quan hóa bộ dữ liệu sử dụng Gephi. Đầu tiên, các nút có ít kết nối đã được lọc ra để cải thiện hiệu suất. Kích thước của các nút còn lại phụ thuộc vào số lượng kết nối của chúng, & màu sắc của chúng biểu thị loại mà chúng thuộc về. Cuối cùng, 2 bố cục đã được áp dụng: Fruchterman-Reingold & ForceAtlas2.

Can import `Facebook Page-Page` dataset same way we did for `Cora`:

1. Import `FacebookPagePage` class from PyTorch Geometric.
2. Store graph in a dedicated `data` variable.
3. Print information about dataset in general.
4. Unlike `Cora, Facebook Page-Page` does not have training, evaluation, & test masks by default. Can arbitrary create masks with `range()` function.

Alternatively, PyTorch Geometric offers a transform function to calculate random masks when dataset is loaded:

```
import torch_geometric.transforms as T
dataset = Planetoid(root=".", name="Cora")
data = dataset[0]
```

1st output confirms number of nodes & classes we saw in description of dataset. 2nd output tells: this graph has `self` loops: some pages are connected to themselves. This is surprising but, in practice, it will not matter.

– Có thể nhập tập dữ liệu `Facebook Page-Page` theo cách chúng ta đã làm với `Cora`:

1. Nhập lớp `FacebookPagePage` từ PyTorch Geometric.
2. Lưu trữ đồ thị trong 1 biến `data` chuyên dụng.
3. In thông tin về tập dữ liệu nói chung.
4. Không giống như `Cora, Facebook Page-Page` mặc định không có mặt nạ huấn luyện, đánh giá, & kiểm tra. Có thể tạo mặt nạ tùy ý bằng hàm `range()`.

Ngoài ra, PyTorch Geometric cung cấp 1 hàm biến đổi để tính toán mặt nạ ngẫu nhiên khi tập dữ liệu được tải:

```
import torch_geometric.transforms as T
dataset = Planetoid(root=".", name="Cora")
data = dataset[0]
```

Đầu ra đầu tiên xác nhận số lượng nút & lớp mà chúng ta đã thấy trong phần mô tả tập dữ liệu. Đầu ra thứ 2 cho biết: đồ thị này có các vòng lặp `self`: 1 số trang được kết nối với chính chúng. Điều này có vẻ đáng ngạc nhiên, nhưng trên thực tế, nó không quan trọng. [NQBH: This code misses some parts?]

These are 2 graph datasets used in next sect, to compare performance of a Vanilla Neural Network to performance of our 1st GNN. Implement them step by step.

– Đây là 2 bộ dữ liệu đồ thị được sử dụng trong phần tiếp theo để so sánh hiệu suất của Mạng nơ-ron nhân tạo Vanilla với hiệu suất của Mạng nơ-ron nhân tạo (GNN) đầu tiên của chúng tôi. Hãy triển khai chúng từng bước một.

◦ Classifying nodes with vanilla neural networks. Compared to Zachary's Karate Club, these 2 datasets include a new type of information: node features. They provide additional information about nodes in a graph, e.g. a user's age, gender, or interests in a social network. In a vanilla neural network (also called *multilayer perceptron*), these embeddings are directly used in model to perform downstream tasks e.g. node classification.

– Phân loại các nút bằng mạng nơ-ron vanilla. So với Zachary's Karate Club, 2 tập dữ liệu này bao gồm 1 loại thông tin mới: đặc điểm nút. Chúng cung cấp thêm thông tin về các nút trong đồ thị, ví dụ: tuổi, giới tính hoặc sở thích của người dùng trên mạng xã hội. Trong mạng nơ-ron vanilla (còn gọi là perceptron đa lớp), các nhúng này được sử dụng trực tiếp trong mô hình để thực hiện các tác vụ hạ nguồn, ví dụ: phân loại nút.

In this sect, consider node features as a regular tabular dataset. Train a simple neural network on this dataset to classify our nodes. Note this architecture does not take into account topology of network. Try to fix this issue in next sect & compare our results. Tabular dataset of node features can be easily accessed through `data` object we created. 1st, would like to convert this object into a regular pandas DataFrame by merging `data.x` (containing node features) & `data.y` (containing class label of each node among 7 classes). In following, use `Cora` dataset:

```
import pandas as pd
df_x = pd.DataFrame(data.x.numpy())
df_x['label'] = pd.DataFrame(data.y)
```

This gives us following dataset Fig. 5.3: Tabular representation of Cora dataset (without topological information).

– Trong phần này, hãy xem xét các đặc trưng nút như 1 tập dữ liệu dạng bảng thông thường. Hãy huấn luyện 1 mạng nơ-ron đơn giản trên tập dữ liệu này để phân loại các nút của chúng ta. Lưu ý rằng kiến trúc này không tính đến cấu trúc liên kết của mạng. Hãy cố gắng khắc phục vấn đề này trong phần tiếp theo & so sánh kết quả của chúng ta. Tập dữ liệu dạng bảng của các đặc trưng nút có thể dễ dàng truy cập thông qua đối tượng `data` mà chúng ta đã tạo. Trước tiên, tôi muốn chuyển đổi đối tượng này thành 1 Pandas DataFrame thông thường bằng cách hợp nhất `data.x` (chứa các đặc trưng nút) & `data.y` (chứa nhãn lớp của mỗi nút trong số 7 lớp). Trong phần sau, hãy sử dụng tập dữ liệu `Cora`:

```
import pandas as pd
df_x = pd.DataFrame(data.x.numpy())
df_x['label'] = pd.DataFrame(data.y)
```

Điều này cho chúng ta tập dữ liệu sau Hình 5.3: Biểu diễn dạng bảng của tập dữ liệu Cora (không có thông tin cấu trúc liên kết).

If familiar with ML, probability recognize a typical dataset with data & labels. Can develop a simple *Multilayer Perceptron (MP)* & train it on `data.x` with labels provided by `data.y`.

– Nếu đã quen thuộc với ML, hãy xác suất nhận dạng 1 tập dữ liệu điển hình với dữ liệu & nhãn. Có thể phát triển 1 *Multilayer Perceptron (MP)* đơn giản & huấn luyện nó trên `data.x` với nhãn được cung cấp bởi `data.y`.

Create our own MLP class with 4 methods:

1. `__init__()` to initialize an instance
2. `forward()` to perform forward pass
3. `fit()` to train model
4. `test()` to evaluate it

Before can train model, must define main metric. There are several metrics for multiclass classification problems: accuracy, F1 score, *Area Under Receiver Operating Characteristic Curve (ROC AUC)* score, & so on. For this work, implement a simple accuracy, which is defined as fraction of correct predictions. It is not best metric for multiclass classification, but simpler to understand. Feel free to replace it with your metric of choice:

```
# define metric as the fraction of correct predictions
def accuracy(y_pred, y_true):
    return torch.sum(y_pred == y_true) / len(y_true)
```

Can start actual implementation. Do not need PyTorch Geometric to implement MLP in this sect. Everything can be done in regular PyTorch with following steps:

1. Import required classes from PyTorch.
2. Create a new class called `MLP`, which will inherit all methods & properties from `torch.nn.Module`.

p. 75+++

- Classifying nodes with vanilla graph neural networks.

- 6. Introducing Graph Convolutional Networks.
- 7. Graph Attention Networks.

PART 3: ADVANCED TECHNIQUES.

- 8. Scaling Up Graph Neural Networks with GraphSAGE.
- 9. Defining Expressiveness for Graph Classification.
- 10. Predicting Links with Graph Neural Networks.
- 11. Generating Graphs Using Graph Neural Networks.
- 12. Learning from Heterogeneous Graphs.
- 13. Temporal Graph Neural Networks.
- 14. Explaining Graph Neural Networks.

PART 4: APPLICATIONS.

- 15. Forecasting Traffic Using A3T-GCN.
- 16. Detecting Anomalies Using Heterogeneous GNNs.
- 17. Building a Recommender System Using LightGCN.
- 18. Unlocking the Potential of Graph Neural Networks for Real-World Applications.

1.3 Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao. Graph Neural Networks: Foundations, Frontiers, & Applications. 2022

- Foreword. "1st comprehensive book covering full spectrum of a young, fast-growing research field, GNNs, written by authoritative authors!" – Jiawei Han (Michael Aiken Chair Professor at University of Illinois at Urbana-Champaign, ACM Fellow & IEEE Fellow)

  – "Cuốn sách toàn diện đầu tiên bao quát toàn bộ lĩnh vực nghiên cứu trẻ, phát triển nhanh chóng, GNN, được viết bởi các tác giả có thẩm quyền!" – Jiawei Han (Giáo sư Michael Aiken tại Đại học Illinois ở Urbana-Champaign, Nghiên cứu viên ACM & Nghiên cứu viên IEEE)

  "This book presents a comprehensive & timely survey on graph representation learning. Edited & contributed by best group of experts in this area, this book is a must-read for students, researchers & practitioners who want to learn anything about GNNs." – Heung-Yeung "Harry" Shum (Former Executive Vice President for Technology & Research at Microsoft Research, ACM Fellow, IEEE Fellow, FREng)

  – "Cuốn sách này trình bày 1 khảo sát toàn diện & kịp thời về học biểu diễn đồ thị. Được biên tập & đóng góp bởi nhóm chuyên gia hàng đầu trong lĩnh vực này, cuốn sách này là tài liệu không thể bỏ qua cho sinh viên, nhà nghiên cứu & những người thực hành muốn tìm hiểu bất cứ điều gì về GNN." – Heung-Yeung "Harry" Shum (Cựu Phó Chủ tịch Điều hành Công nghệ & Nghiên cứu tại Microsoft Research, Học giả ACM, Học giả IEEE, FREng)

  "As new frontier of DL, GNNs offer great potential to combine probabilistic learning & symbolic reasoning, & bridge knowledge-driven & data-driven paradigms, nurturing development of 3rd-generation AI. This book provides a comprehensive & insightful introduction to GNN, ranging from foundations to frontiers, from algorithms to applications. It is a valuable resource for any scientist, engineer & student who wants to get into this exciting field." – Bo Zhang (Member of Chinese Academy of Science, Professor at Tsinghua University)

  – "Là 1 lĩnh vực mới của DL, mạng lưới thần kinh nhân tạo (GNN) mang lại tiềm năng to lớn trong việc kết hợp học xác suất & suy luận biểu tượng, & kết nối các mô hình dựa trên kiến thức & dựa trên dữ liệu, nuôi dưỡng sự phát triển của AI thế hệ thứ 3. Cuốn sách này cung cấp 1 giới thiệu toàn diện & sâu sắc về GNN, từ nền tảng đến lĩnh vực, từ thuật toán đến ứng dụng. Đây là 1 nguồn tài nguyên quý giá cho bất kỳ nhà khoa học, kỹ sư & sinh viên nào muốn bước vào lĩnh vực thú vị này." – Bo Zhang (Viện Hàn lâm Khoa học Trung Quốc, Giáo sư tại Đại học Thanh Hoa)

  "GNNs are 1 of hottest areas of ML & this book is a wonderful in-depth resource covering a broad range of topics & applications of graph representation learning." – Jure Leskovec (Associate Professor at Stanford University, & investigator at Chan Zuckerberg Biohub).

  – "GNN là 1 trong những lĩnh vực hấp dẫn nhất của ML & cuốn sách này là nguồn tài nguyên chuyên sâu tuyệt vời bao gồm nhiều chủ đề & ứng dụng của việc học biểu diễn đồ thị." – Jure Leskovec (Phó giáo sư tại Đại học Stanford & nhà nghiên cứu tại Chan Zuckerberg Biohub).

  "GNNs are an emerging ML model that is already taking scientific & industrial world by storm. Time is perfect to get in on action – & this book is a great resource for newcomers & reasoned practitioners alike! Its chaps are very carefully written by many of thought leaders at forefront of area." – Petar Veličković (Senior Research Scientist, DeepMind)

  – "GNN là 1 mô hình ML mới nổi đang gây sốt trong giới khoa học & công nghiệp. Đã đến lúc bắt tay vào hành động – & cuốn sách này là 1 nguồn tài nguyên tuyệt vời cho cả người mới bắt đầu & những người thực hành có lý trí! Các bài viết trong đó được viết rất cẩn thận bởi nhiều nhà tư tưởng hàng đầu trong lĩnh vực này." – Petar Veličković (Nhà khoa học nghiên cứu cao cấp, DeepMind)

- Preface. Field of GNNs has seen rapid & incredible strides over recent years. GNNs, also known as DL on graphs, graph representation learning, or geometric DL, have become 1 of fastest-growing research topics in ML, especially DL. This wave of research at intersection of graph theory & DL has also influenced other fields of science, including recommendation systems, computer vision, NLP, inductive logic programming, program synthesis, software mining, automated planning, cybersecurity, & intelligent transportation.

  – Lĩnh vực GNN đã chứng kiến những bước tiến nhanh chóng & đáng kinh ngạc trong những năm gần đây. GNN, còn được gọi là DL trên đồ thị, học biểu diễn đồ thị, hoặc DL hình học, đã trở thành 1 trong những chủ đề nghiên cứu phát triển nhanh nhất trong ML, đặc biệt là DL. Làn sóng nghiên cứu giao thoa giữa lý thuyết đồ thị & DL này cũng đã ảnh hưởng đến các lĩnh vực khoa học khác, bao gồm hệ thống đề xuất, thị giác máy tính, NLP, lập trình logic quy nạp, tổng hợp chương trình, khai thác phần mềm, lập kế hoạch tự động, an ninh mạng, & giao thông thông minh.

  Although GNNs have achieved remarkable attention, it still faces many challenges when applying them into other domains, from theoretical understanding of methods to scalability & interprebility in a real system, & from soundness of methodology to empirical performance in an application. However, as field rapidly grows, it has been extremely challenging to gain a global

perspective of developments of GNNs. Therefore, feel urgency to bridge above gap & have a comprehensive book on this fast-growing yet challenging topic, which an benefit a broad audience including advanced undergraduate & graduate students, postdoctoral researchers, lecturers, & industrial practitioners.

– Mặc dù GNN đã thu hút được sự chú ý đáng kể, nhưng nó vẫn gặp nhiều thách thức khi áp dụng vào các lĩnh vực khác, từ hiểu biết lý thuyết về phương pháp đến khả năng mở rộng & khả năng diễn giải trong hệ thống thực, & từ tính vững chắc của phương pháp luận đến hiệu suất thực nghiệm trong ứng dụng. Tuy nhiên, khi lĩnh vực này phát triển nhanh chóng, việc có được cái nhìn toàn cầu về sự phát triển của GNN là vô cùng khó khăn. Do đó, chúng ta cần khẩn trương thu hẹp khoảng cách & có 1 cuốn sách toàn diện về chủ đề đang phát triển nhanh chóng nhưng đầy thách thức này, có thể mang lại lợi ích cho nhiều đối tượng, bao gồm sinh viên đại học cao cấp & sau đại học, nghiên cứu sinh sau tiến sĩ, giảng viên, & những người hành nghề trong ngành.

This book is intended to cover a broad range of topics in GNNs, from foundations to frontiers, & from methodologies to applications. Book is dedicated to introducing fundamental concepts & algorithms of GNNs, new research frontiers of GNNs, & broad & emerging applications with GNNs.

– Cuốn sách này được thiết kế để bao quát 1 loạt các chủ đề về GNN, từ nền tảng đến các lĩnh vực tiên tiến, & từ phương pháp luận đến ứng dụng. Sách dành riêng để giới thiệu các khái niệm cơ bản & thuật toán của GNN, các lĩnh vực nghiên cứu mới của GNN, & các ứng dụng rộng & mới nổi của GNN.

- Book Website & Resources. Website https://graph-neural-networks.github.io provides online preprints & lecture slides of all chaps, also provides pointers to useful material & resources publicly available & relevant to GNNs.
- To Instructors. Book can be used for 1-semester graduate course for graduate students. Though mainly written for students with a background in CS, students with a basic understanding of probability, statistics, graph theory, linear algebra, & ML techniques e.g. DL will find it easily accessible. Some chaps can be skipped or assigned as homework assignments for reviewing purposes if students have knowledge of a chap. Instructors can choose to combine Chaps. 1–3 together as background introduction course at beginning.

  – Sách có thể được sử dụng cho các khóa học sau đại học kéo dài 1 học kỳ dành cho sinh viên cao học. Mặc dù chủ yếu được viết cho sinh viên có nền tảng về Khoa học Máy tính, nhưng sinh viên có kiến thức cơ bản về xác suất, thống kê, lý thuyết đồ thị, đại số tuyến tính, & các kỹ thuật Học máy (ML) như DL sẽ dễ dàng tiếp cận. 1 số chương có thể được bỏ qua hoặc giao làm bài tập về nhà để ôn tập nếu sinh viên đã biết về 1 chương nào đó. Giảng viên có thể chọn kết hợp các Chương 1-3 lại với nhau như 1 khóa học nhập môn nền tảng lúc bắt đầu.

  When course focuses more on foundation & theories of GNNs, instructor can choose to focus more on Chaps. 4–8 while using Chaps. 19–27 to showcase applications, motivations, & limitations. When course focuses more on research frontiers, Chaps. 9–18 can be pivot to organize course. E.g., an instructor can make it an advanced graduate course where students are asked to search & present most recent research papers in each different research frontier. They can also be asked to established their course projects based on applications described in Chaps. 19–27 as well as materials provided on website.

  – Khi khóa học tập trung nhiều hơn vào nền tảng & lý thuyết của GNN, giảng viên có thể chọn tập trung nhiều hơn vào Chương 4-8 trong khi sử dụng Chương 19-27 để trình bày các ứng dụng, động lực, & hạn chế. Khi khóa học tập trung nhiều hơn vào các lĩnh vực nghiên cứu, Chương 9-18 có thể là bước đệm để tổ chức khóa học. Ví dụ: giảng viên có thể biến nó thành 1 khóa học sau đại học nâng cao, trong đó sinh viên được yêu cầu tìm kiếm & trình bày các bài báo nghiên cứu gần đây nhất trong mỗi lĩnh vực nghiên cứu khác nhau. Họ cũng có thể được yêu cầu thiết lập các dự án khóa học dựa trên các ứng dụng được mô tả trong Chương 19-27 cũng như các tài liệu được cung cấp trên trang web.
- To Readers. This book was designed to cover a wide range of topics in field of GNN field, including background, theoretical foundations, methodologies, researcher frontier, & applications. Therefore, it can be treated as a comprehensive handbook for a wide variety of readers e.g. students, researchers, & professionals. Should have some knowledge of concepts & terminology associated with statistics, ML, & graph theory. Some backgrounds of basics have been provided & referenced in 1st 8 chaps. Should better also have knowledge of DL & Some programming experience for easily accessing most of chaps of this book. In particular, should be able to read pseudocode & understand graph structures.

  – Cuốn sách này được thiết kế để bao quát 1 loạt các chủ đề trong lĩnh vực Mạng Lưới Mạng (GNN), bao gồm bối cảnh, nền tảng lý thuyết, phương pháp luận, ranh giới nghiên cứu, & ứng dụng. Do đó, nó có thể được coi là 1 cẩm nang toàn diện cho nhiều đối tượng độc giả, chẳng hạn như sinh viên, nhà nghiên cứu, & chuyên gia. Cần có 1 số kiến thức về các khái niệm & thuật ngữ liên quan đến thống kê, Học máy, & lý thuyết đồ thị. 1 số kiến thức cơ bản đã được cung cấp & tham chiếu trong 8 chương đầu tiên. Tốt hơn hết là nên có kiến thức về Học máy & 1 số kinh nghiệm lập trình để dễ dàng tiếp cận hầu hết các chương của cuốn sách này. Đặc biệt, cần có khả năng đọc mã giả & hiểu các cấu trúc đồ thị.

  Book is well modularized & each chap can be learned in a standalone manner based on individual interests & needs. For those readers who want to have a solid understanding of various techniques & theories of GNNs, can start from Chaps. 4–9. For those who further want to perform in-depth research & advanced related fields, read those chaps of interest among Chaps. 9–18, which provide comprehensive knowledge in most recent research issues, open problems, & research frontiers. For those who want to apply GNNs to benefit specific domains, or aim at finding interesting applications to validate specific GNNs techniques, refer to Chaps. 19–27.

  –Sách được phân chia thành các module rõ ràng & mỗi chương có thể được học độc lập dựa trên sở thích & nhu cầu cá nhân. Đối với những độc giả muốn có hiểu biết vững chắc về các kỹ thuật & lý thuyết khác nhau về GNN, có thể bắt đầu từ Chương 4-9. Đối với những độc giả muốn nghiên cứu sâu hơn & các lĩnh vực liên quan nâng cao, hãy đọc các chương quan tâm trong Chương 9-18, cung cấp kiến thức toàn diện về các vấn đề nghiên cứu gần đây nhất, các vấn đề còn bỏ ngỏ,

& các lĩnh vực nghiên cứu tiên tiến. Đối với những độc giả muốn áp dụng GNN để mang lại lợi ích cho các lĩnh vực cụ thể, hoặc muốn tìm kiếm các ứng dụng thú vị để xác thực các kỹ thuật GNN cụ thể, hãy tham khảo Chương 19-27.

- Terminologies. This chap describes a list of definitions of terminologies related to GNNs used throughout this book.
  - 1. Basic concepts of Graphs.
    1. **Graph.** A graph is composed of a node set & an edge set, where nodes represent entities & edges represent relationship between entities. Nodes & edges form topology structure of graph. Besides graph structure, nodes, edges, &/or whole graph can be associated with rich information represented as node/edge/graph features (also known as attributes or contents).
       – 1 đồ thị bao gồm 1 tập nút & 1 tập cạnh, trong đó các nút biểu diễn các thực thể & các cạnh biểu diễn mối quan hệ giữa các thực thể. Các nút & các cạnh tạo nên cấu trúc tôpô của đồ thị. Bên cạnh cấu trúc đồ thị, các nút, cạnh, &/hoặc toàn bộ đồ thị có thể được liên kết với thông tin phong phú được biểu diễn dưới dạng các đặc trưng nút/cạnh/đồ thị (còn được gọi là thuộc tính hoặc nội dung).
    2. **Subgraph.** A subgraph is a graph whose set of nodes & set of edges are all subsets of original graph.
       – Đồ thị con là đồ thị có tập hợp các nút & tập hợp các cạnh đều là tập con của đồ thị gốc.
    3. **Centrality.** A centrality is a measurement of importance of nodes in graph. Basic assumption of centrality: a node is thought to be important if many other important nodes also connect to it. Common centrality measurements include degree centrality, eigenvector centrality, betweenness centrality, & closeness centrality.
       – Độ trung tâm là phép đo tầm quan trọng của các nút trong đồ thị. Giả định cơ bản về độ trung tâm: 1 nút được coi là quan trọng nếu nhiều nút quan trọng khác cũng kết nối với nó. Các phép đo độ trung tâm phổ biến bao gồm độ trung tâm bậc, độ trung tâm vectơ riêng, độ trung tâm giữa, độ trung tâm & độ gần.
    4. **Neighborhood.** Neighborhood of a node generally refers to other nodes that are close to it. E.g., k-order neighborhood of a node, also called k-step neighborhood, denotes a set of other nodes in which shortest path distance between these nodes & central node is no larger than k.
       – Lân cận của 1 nút thường đề cập đến các nút khác gần nó. E.g., lân cận bậc k của 1 nút, còn được gọi là lân cận bậc k, biểu thị 1 tập hợp các nút khác có khoảng cách đường dẫn ngắn nhất giữa các nút này & nút trung tâm không lớn hơn k.
    5. **Community structure.** A community refers to a group of nodes that are densely connected internally & less densely connected externally.
       – **Cấu trúc cộng đồng.** Cộng đồng đề cập đến 1 nhóm các nút được kết nối dày đặc bên trong & kết nối ít dày đặc hơn bên ngoài.
    6. **Graph sampling.** Graph sampling is a technique to pick a subset of nodes &/or edges from original graph. Graph sampling can be applied to train ML models on large-scale graphs while preventing severe scalability issues.
       – Lấy mẫu đồ thị là 1 kỹ thuật để chọn 1 tập hợp con các nút &/hoặc cạnh từ đồ thị gốc. Lấy mẫu đồ thị có thể được áp dụng để huấn luyện các mô hình ML trên đồ thị quy mô lớn, đồng thời ngăn ngừa các vấn đề nghiêm trọng về khả năng mở rộng.
    7. **Heterogeneous Graphs.** Graphs are called heterogeneous if nodes &/or edges of graph are from different types. A typical example of heteronomous graphs is knowledge graphs where edges are composed of different types.
       – **Đồ thị không đồng nhất.** Đồ thị được gọi là không đồng nhất nếu các nút &/hoặc các cạnh của đồ thị thuộc các loại khác nhau. 1 ví dụ điển hình của đồ thị không đồng nhất là đồ thị tri thức, trong đó các cạnh được tạo thành từ các loại khác nhau.
    8. **Hypergraphs.** Hypergraphs are generalizations of graphs in which an edge can join any number of nodes.
       – **Siêu đồ thị.** Siêu đồ thị là dạng tổng quát của đồ thị trong đó 1 cạnh có thể nối với bất kỳ số lượng nút nào.
    9. **Random Graph.** Random graph generally aims to model probability distributions over graphs that observed graphs are generated from. Most basic & well-suited random graph model, known as Erdos–Renyi model, assumes: node set is fixed & each edge is identically & independently generated.
       – **Đồ thị ngẫu nhiên.** Đồ thị ngẫu nhiên thường nhằm mục đích mô hình hóa phân phối xác suất trên các đồ thị mà đồ thị quan sát được tạo ra. Mô hình đồ thị ngẫu nhiên cơ bản nhất, được gọi là mô hình Erdos-Renyi, giả định: tập hợp nút là cố định & mỗi cạnh được tạo ra độc lập & giống hệt nhau.
    10. **Dynamic Graph.** Dynamic graph refers to when at least 1 component of graph data changes over time, e.g., adding or deleting nodes, adding or deleting edges, changing edges weights or changing node attributes, etc. If graphs are not dynamic, refer to them as static graphs.
       – **Đồ thị động.** Đồ thị động đề cập đến trường hợp ít nhất 1 thành phần của dữ liệu đồ thị thay đổi theo thời gian, e.g.: thêm hoặc xóa các nút, thêm hoặc xóa các cạnh, thay đổi trọng số cạnh hoặc thay đổi các thuộc tính của nút, v.v. Nếu đồ thị không động, gọi chúng là đồ thị tĩnh.
  - 2. ML on Graphs.
    1. **Spectral Graph Theory.** Spectral graph theory analyzes matrices associated with graph e.g. its adjacency matrix or Laplacian matrix using tools of linear algebra e.g. studying eigenvalues & eigenvectors of matrix.
       – **Lý thuyết đồ thị phổ.** Lý thuyết đồ thị phổ phân tích các ma trận liên quan đến đồ thị, e.g. ma trận kề hoặc ma trận Laplacian bằng các công cụ của đại số tuyến tính, e.g. nghiên cứu các giá trị riêng & vectơ riêng của ma trận.

2. **Graph Signal Processing.** Graph Signal Processing (GSP) aims to develop tools for processing signals defined on graphs. A graph signal refers to a finite collection of data samples with 1 sample at each node in graph.
   – **Xử lý Tín hiệu Đồ thị.** Xử lý Tín hiệu Đồ thị (GSP) nhằm mục đích phát triển các công cụ để xử lý các tín hiệu được xác định trên đồ thị. Tín hiệu đồ thị là 1 tập hợp hữu hạn các mẫu dữ liệu với 1 mẫu tại mỗi nút trong đồ thị.
3. **Node-level Tasks.** Node-level tasks refer to ML tasks associated with individual nodes in graph. Typical examples of node-level tasks include node classification & node regression.
   – **Nhiệm vụ cấp nút.** Nhiệm vụ cấp nút đề cập đến các nhiệm vụ học máy (ML) được liên kết với từng nút trong đồ thị. Các ví dụ điển hình về nhiệm vụ cấp nút bao gồm phân loại nút & hồi quy nút.
4. **Edge-level Tasks.** Edge-level tasks refer to ML tasks associated with a pair of nodes in graph. A typical example of an edge-level task in link prediction.
   – **Nhiệm vụ cấp biên.** Nhiệm vụ cấp biên đề cập đến các nhiệm vụ ML liên quan đến 1 cặp nút trong đồ thị. 1 ví dụ điển hình về nhiệm vụ cấp biên trong dự đoán liên kết.
5. **Graph-level Tasks.** Graph-level tasks refer to ML tasks associated with whole graph. Typical examples of graph-level tasks include graph classification & graph property prediction.
   – **Nhiệm vụ cấp độ đồ thị.** Nhiệm vụ cấp độ đồ thị đề cập đến các nhiệm vụ học máy (ML) liên quan đến toàn bộ đồ thị. Các ví dụ điển hình của nhiệm vụ cấp độ đồ thị bao gồm phân loại đồ thị & dự đoán thuộc tính đồ thị.
6. **Transductive & Inductive Learning.** Transductive learning refers to targeted instances e.g. nodes or edges are observed at training time (though labels of targeted instances remain unknown) & inductive learning aims to learn model which is generalizable to unobserved instances.
   – **Học chuyển tiếp & Học quy nạp.** Học chuyển tiếp đề cập đến các trường hợp mục tiêu, e.g. các nút hoặc cạnh được quan sát tại thời điểm đào tạo (mặc dù nhãn của các trường hợp mục tiêu vẫn chưa được biết) & học quy nạp nhằm mục đích học mô hình có thể khái quát hóa cho các trường hợp chưa được quan sát.

○ 3. GNNs.

∗ **Network embedding.** Goal of network embedding: represent each node in graph as a low-dimensional vector so that useful information e.g. graph structures & some properties of graph is preserved in embedding vectors. Network embedding is also referred to as graph embedding & node representation learning.
   – **Nhúng mạng.** Mục tiêu của nhúng mạng: biểu diễn mỗi nút trong đồ thị dưới dạng 1 vectơ ít chiều để thông tin hữu ích, e.g. cấu trúc đồ thị & 1 số thuộc tính của đồ thị, được bảo toàn trong các vectơ nhúng. Nhúng mạng còn được gọi là học biểu diễn nút & nhúng đồ thị.

∗ **Graph Neural Network.** GNN refers to any neural network working on graph data.
   – **Mạng nơ-ron đồ thị.** GNN đề cập đến bất kỳ mạng nơ-ron nào hoạt động trên dữ liệu đồ thị.

∗ **Graph Convolutional Network.** Graph convolutional network usually refers to a specific GNN proposed by Kipf & Welling. It is occasionally used as a synonym for GNN, i.e., referring to any neural network working on graph data, in some literature.
   – **Mạng Tích chập Đồ thị.** Mạng tích chập đồ thị thường đề cập đến 1 GNN cụ thể do Kipf & Welling đề xuất. Đôi khi, nó được dùng như 1 từ đồng nghĩa với GNN, i.e., đề cập đến bất kỳ mạng nơ-ron nào hoạt động trên dữ liệu đồ thị, trong 1 số tài liệu.

∗ **Message-Passing.** Message-passing is a framework of GNNs in which key step: pass messages between different nodes based on graph structures in each neural network layer. Most widely adopted formulation, usually denoted as message-passing neural networks, is to only pass messages between nodes that are directly connected Gilmer et al (2017). Message passing functions are also called *graph filters & graph convolutions* in some literature.
   – **Truyền thông điệp.** Truyền thông điệp là 1 khuôn khổ của mạng nơ-ron nhân tạo (GNN), trong đó bước quan trọng nhất là truyền thông điệp giữa các nút khác nhau dựa trên cấu trúc đồ thị trong mỗi lớp mạng nơ-ron. Công thức được áp dụng rộng rãi nhất, thường được gọi là mạng nơ-ron truyền thông điệp, là chỉ truyền thông điệp giữa các nút được kết nối trực tiếp (Gilmer et al. (2017)). Các hàm truyền thông điệp còn được gọi là bộ lọc đồ thị & tích chập đồ thị trong 1 số tài liệu.

∗ **Readout.** Readout refers to functions that summarize information of individual nodes to form more high-level information e.g. forming a subgraph/supergraph or obtaining representations of entire graph. Readout is also called pooling & graph coarsening in some literature.
   – **Đọc ra.** Đọc ra đề cập đến các hàm tóm tắt thông tin của từng nút riêng lẻ để tạo thành thông tin cấp cao hơn, e.g.: tạo 1 siêu đồ thị hoặc thu thập biểu diễn của toàn bộ đồ thị. Đọc ra còn được gọi là gộp & làm thô đồ thị trong 1 số tài liệu.

∗ **Graph Adversarial Attack.** Graph adversarial attacks aim to generate worst-case perturbations by manipulating graph structure &/or node features so that performance of some models are downgraded. Graph adversarial attacks can be categorized based on attacker's goals, capabilities, & accessible knowledge.
   – **Tấn công đối kháng đồ thị.** Các cuộc tấn công đối kháng đồ thị nhằm mục đích tạo ra nhiễu loạn trường hợp xấu nhất bằng cách thao túng cấu trúc đồ thị &/hoặc các đặc điểm nút để làm giảm hiệu suất của 1 số mô hình. Các cuộc tấn công đối kháng đồ thị có thể được phân loại dựa trên mục tiêu, khả năng & kiến thức có thể tiếp cận của kẻ tấn công.

∗ **Robustness certificates.** Methods providing formal guarantees prediction of a GNN is not affected even when perturbations are performed based on a certain perturbation model.
   – **Chứng chỉ độ tin cậy.** Các phương pháp cung cấp bảo đảm chính thức cho khả năng dự đoán GNN không bị ảnh hưởng ngay cả khi nhiễu loạn được thực hiện dựa trên 1 mô hình nhiễu loạn nhất định.

PART I. INTRODUCTION.

- 1. Liang Zhao, Lingfei Wu, Peng Cui, Jian Pei. Representation Learning. In this chap, 1st describe what representation learning is & why need representation learning. Among various ways of learning representations, this chap focuses on DL methods: those formed by composition of multiple nonlinear transformations, with goal of resulting in more abstract & ultimately more useful representations. Summarize representation learning techniques in different domains, focusing on unique challenges & models for different data types including images, natural languages, speech signals & networks.

  – Trong chương này, trước tiên, chúng ta sẽ mô tả học biểu diễn là gì & tại sao cần học biểu diễn. Trong số các phương pháp học biểu diễn khác nhau, chương này tập trung vào các phương pháp DL: những phương pháp được hình thành từ việc kết hợp nhiều phép biến đổi phi tuyến tính, với mục tiêu tạo ra các biểu diễn trừu tượng hơn & cuối cùng là hữu ích hơn. Tóm tắt các kỹ thuật học biểu diễn trong các lĩnh vực khác nhau, tập trung vào các thách thức & mô hình độc đáo cho các loại dữ liệu khác nhau bao gồm hình ảnh, ngôn ngữ tự nhiên, tín hiệu giọng nói & mạng.

  - 1.1. Representation Learning: An Introduction. Effectiveness of ML techniques heavily relies on not only design of algorithms themselves, but also a good representation (feature set) of data. Ineffective data representations that lack some important information or contains incorrect or huge redundant information could lead to poor performance of algorithm in dealing with different tasks. Goal of representation learning: extract sufficient but minimal information from data. Traditionally, this can be achieved via human efforts based on prior knowledge & domain expertise on data & tasks, which is also named as feature engineering. In deploying ML & many other AI algorithms, historically a large portion of human efforts goes into design of preprocessing pipelines & data transformations. More specifically, feature engineering is a way to take advantage of human ingenuity & prior knowledge in hope to extract & organize discriminative information from data for ML tasks. E.g., political scientists may be asked to define a keyword list as features of social-media text classifiers for detecting those texts on societal events. For speech transcription recognition, one may choose to extract features from raw sound waves by operations including Fourier transformations. Although feature engineering is widely adopted over years, its drawbacks are also salient, including:

    1. Intensive labors from domain experts are usually needed. This is because feature engineering may require tight & extensive collaboration between model developers & domain experts.
    2. Incomplete & biased feature extraction. Specically, capacity & discriminative power of extracted features are limited by knowledge of different domain experts. Moreover, in many domains that human beings have limited knowledge, what features to extract itself is an open question to domain experts, e.g. cancer early prediction.

    In order to avoid these drawbacks, ML algorithms less dependent on feature engineering has been a highly desired goal in ML & AI domains, so that novel applications could be constructed faster & hopefully addressed more effectively.

    – Học Biểu Diễn: Giới thiệu. Hiệu quả của các kỹ thuật ML không chỉ phụ thuộc vào thiết kế thuật toán mà còn vào việc biểu diễn dữ liệu (bộ đặc trưng) tốt. Việc biểu diễn dữ liệu kém hiệu quả, thiếu 1 số thông tin quan trọng hoặc chứa thông tin không chính xác hoặc quá nhiều thông tin dư thừa, có thể dẫn đến hiệu suất thuật toán kém khi xử lý các tác vụ khác nhau. Mục tiêu của học biểu diễn: trích xuất đủ nhưng tối thiểu thông tin từ dữ liệu. Theo truyền thống, điều này có thể đạt được thông qua nỗ lực của con người dựa trên kiến thức & chuyên môn về dữ liệu & tác vụ, còn được gọi là kỹ thuật đặc trưng. Khi triển khai ML & nhiều thuật toán AI khác, trước đây, phần lớn nỗ lực của con người được dành cho việc thiết kế các đường ống tiền xử lý & chuyển đổi dữ liệu. Cụ thể hơn, kỹ thuật đặc trưng là 1 cách tận dụng sự khéo léo của con người & kiến thức sẵn có với hy vọng trích xuất & tổ chức thông tin phân biệt từ dữ liệu cho các tác vụ ML. Ví dụ: các nhà khoa học chính trị có thể được yêu cầu định nghĩa 1 danh sách từ khóa là các đặc trưng của bộ phân loại văn bản trên mạng xã hội để phát hiện các văn bản đó về các sự kiện xã hội. Đối với nhận dạng phiên âm giọng nói, người ta có thể chọn trích xuất các đặc trưng từ sóng âm thô bằng các phép toán bao gồm biến đổi Fourier. Mặc dù kỹ thuật thiết kế đặc trưng đã được áp dụng rộng rãi trong nhiều năm, nhưng nó cũng có những nhược điểm nổi bật, bao gồm:

    1. Thường cần rất nhiều công sức từ các chuyên gia trong lĩnh vực. Điều này là do kỹ thuật thiết kế đặc trưng có thể đòi hỏi sự hợp tác chặt chẽ & sâu rộng giữa các nhà phát triển mô hình & chuyên gia trong lĩnh vực.
    2. Trích xuất đặc trưng không đầy đủ & thiên vị. Cụ thể, khả năng & phân biệt của các đặc trưng được trích xuất bị hạn chế bởi kiến thức của các chuyên gia trong lĩnh vực khác nhau. Hơn nữa, trong nhiều lĩnh vực mà con người có kiến thức hạn chế, việc trích xuất đặc trưng nào là 1 câu hỏi mở đối với các chuyên gia trong lĩnh vực, e.g. dự đoán sớm ung thư.

    Để tránh những nhược điểm này, các thuật toán ML ít phụ thuộc vào kỹ thuật thiết kế đặc trưng đã trở thành 1 mục tiêu rất được mong đợi trong lĩnh vực ML & AI, để các ứng dụng mới có thể được xây dựng nhanh hơn & hy vọng được giải quyết hiệu quả hơn.

    Techniques of representation learning witness development from traditional representation learning techniques to more advanced ones. Traditional methods belong to "shallow" models & aim to learn transformations of data that make it easier to extract useful information when building classifiers or other predictors, e.g. Principal Component Analysis (PCA) (Wold et al, 1987), Gaussian Markov random field (GMRF) (Rue & Held, 2005), & Locality Preserving Projections (LPP) (He & Niyogi, 2004). DL-based representation learning is formed by composition of multiple nonlinear transformations, with goal of yielding more abstract & ultimately more useful representations. In light of introducing more recent advancements & sticking to major topic of this book, here we majorly focus on DL-based representation learning, which can be categorized into several types:

    1. Supervised learning, where a large number of labeled data are needed for training of DL models. Given well-trained networks, output before last fully-connected layers is always utilized as final representation of input data.

2. Unsupervised learning (including self-supervised learning), which facilitates analysis of input data without corresponding labels & aims to learn underlying inherent structure or distribution of data. Pre-tasks are utilized to explore supervision information from large amounts of unlabeled data. Based on this constructed supervision information, deep neural networks DNNs are trained to extract meaningful representations for future downstream tasks.
3. Transfer learning, which involves methods that utilize any knowledge resource (i.e., data, model, labels, etc.) to increase model learning & generalization for target task. Transfer learning encompasses different scenarios including multi-task learning (MTL), model adaptation, knowledge transfer, co-variance shift, etc.

There are also other important representation learning methods e.g. reinforcement learning, few-shot learning, & disentangled representation learning.

– Các kỹ thuật học biểu diễn chứng kiến sự phát triển từ các kỹ thuật học biểu diễn truyền thống đến các kỹ thuật tiên tiến hơn. Các phương pháp truyền thống thuộc về các mô hình "nông" & hướng đến việc học các phép biến đổi dữ liệu giúp trích xuất thông tin hữu ích dễ dàng hơn khi xây dựng các bộ phân loại hoặc các yếu tố dự đoán khác, e.g.: Phân tích Thành phần Chính (PCA) (Wold & cộng sự, 1987), Trường ngẫu nhiên Gauss Markov (GMRF) (Rue & Held, 2005), & Phép chiếu Bảo toàn Vị trí (LPP) (He & Niyogi, 2004). Học biểu diễn dựa trên DL được hình thành bằng cách kết hợp nhiều phép biến đổi phi tuyến tính, với mục tiêu tạo ra các biểu diễn trừu tượng hơn & cuối cùng là hữu ích hơn. Nhằm giới thiệu những tiến bộ gần đây & bám sát chủ đề chính của cuốn sách này, ở đây chúng tôi chủ yếu tập trung vào học biểu diễn dựa trên DL, có thể được phân loại thành 1 số loại:

1. Học có giám sát, trong đó cần 1 lượng lớn dữ liệu được gắn nhãn để huấn luyện các mô hình DL. Với các mạng được đào tạo tốt, đầu ra trước các lớp kết nối đầy đủ cuối cùng luôn được sử dụng làm biểu diễn cuối cùng của dữ liệu đầu vào.
2. Học không giám sát (bao gồm học tự giám sát), giúp phân tích dữ liệu đầu vào mà không cần nhãn tương ứng & nhằm mục đích tìm hiểu cấu trúc hoặc phân phối dữ liệu vốn có bên dưới. Các tác vụ tiền nhiệm được sử dụng để khám phá thông tin giám sát từ 1 lượng lớn dữ liệu chưa được gắn nhãn. Dựa trên thông tin giám sát được xây dựng này, các mạng nơ-ron sâu (DNN) được đào tạo để trích xuất các biểu diễn có ý nghĩa cho các tác vụ hạ nguồn trong tương lai.
3. Học chuyển giao, bao gồm các phương pháp sử dụng bất kỳ nguồn kiến thức nào (i.e., dữ liệu, mô hình, nhãn, v.v.) để tăng cường học mô hình & khái quát hóa cho tác vụ mục tiêu. Học chuyển giao bao gồm các kịch bản khác nhau, bao gồm học đa tác vụ (MTL), thích ứng mô hình, chuyển giao kiến thức, dịch chuyển hiệp phương sai, v.v.

Ngoài ra còn có các phương pháp học biểu diễn quan trọng khác, e.g. học tăng cường, học ít lần, & học biểu diễn không rối.

Important to define what is a good representation. As def by Bengio2008, representation learning is about learning (underlying) features of data that make it easier to extract useful information when building classifiers or other predictors. Thus evaluation of a learned representation is closely related to its performance on downstream tasks. E.g., in data generation task based on a generative model, a good representation is often the one that captures posterior distribution of underlying explanatory factors for observed input. While for a prediction task, a good representation is the one that captures minimal but sufficient information of input data to correctly predict target label. Besides evaluation from perspective of downstream tasks, there are also some general properties that good representations may hold, e.g. smoothness, linearity, capturing multiple explanatory & casual factors, holding shared factors across different tasks & simple factor dependencies.

– Điều quan trọng là phải xác định thế nào là 1 biểu diễn tốt. Theo định nghĩa của Bengio2008, học biểu diễn là về việc học các đặc điểm (nền tảng) của dữ liệu giúp trích xuất thông tin hữu ích dễ dàng hơn khi xây dựng các bộ phân loại hoặc các yếu tố dự đoán khác. Do đó, việc đánh giá 1 biểu diễn đã học có liên quan chặt chẽ đến hiệu suất của nó trong các tác vụ hạ lưu. Ví dụ: trong tác vụ tạo dữ liệu dựa trên mô hình sinh, 1 biểu diễn tốt thường là biểu diễn nắm bắt được phân phối sau của các yếu tố giải thích cơ bản đối với đầu vào được quan sát. Trong khi đối với tác vụ dự đoán, 1 biểu diễn tốt là biểu diễn nắm bắt được thông tin tối thiểu nhưng đủ về dữ liệu đầu vào để dự đoán chính xác nhãn mục tiêu. Bên cạnh việc đánh giá từ góc độ của các tác vụ hạ lưu, cũng có 1 số thuộc tính chung mà các biểu diễn tốt có thể có, e.g.: độ mượt, tính tuyến tính, nắm bắt nhiều yếu tố giải thích & ngẫu nhiên, giữ các yếu tố được chia sẻ trên các tác vụ khác nhau & phụ thuộc yếu tố đơn giản.

- 1.2. Representation Learning in Different Areas. In this sect, summarize development of representation learning on 4 different representative areas: (1) image processing; (2) speech recognition; (3) NLP; & (4) network analysis. For representation learning in each research area, consider some of fundamental questions that have been driving research in this area. Specifically, what makes 1 representation better than another, & how should we compute its representation? Why is representation learning important in that area? Also, what are appropriate objectives for learning good representations? Also introduce relevant typical methods & their development from perspective of 3 main categories: supervised representation learning, unsupervised learning & transfer learning, resp.

  – Học Biểu Diễn trong Các Lĩnh Vực Khác Nhau. Trong phần này, tóm tắt sự phát triển của học biểu diễn trên 4 lĩnh vực đại diện khác nhau: (1) xử lý ảnh; (2) nhận dạng giọng nói; (3) NLP; & (4) phân tích mạng. Đối với học biểu diễn trong mỗi lĩnh vực nghiên cứu, hãy xem xét 1 số câu hỏi cơ bản đã thúc đẩy nghiên cứu trong lĩnh vực này. Cụ thể, điều gì làm cho 1 biểu diễn tốt hơn biểu diễn khác, & chúng ta nên tính toán biểu diễn của nó như thế nào? Tại sao học biểu diễn lại quan trọng trong lĩnh vực đó? Ngoài ra, mục tiêu phù hợp để học các biểu diễn tốt là gì? Đồng thời giới thiệu các phương pháp điển hình có liên quan & sự phát triển của chúng theo góc nhìn của 3 loại chính: học biểu diễn có giám sát, học không giám sát & học chuyển giao, tương ứng.

  * 1.2.1. Representation Learning for Image Processing. Image representation learning is a fundamental problem in understanding semantics of various visual data, e.g. photographs, medical images, document scans, & video streams. Normally, goal of image representation learning for image processing: bridge semantic gap between pixel data & semantics of images.

Successful achievements of image representation learning have empowered many real-world problems, including but not limited to image search, facial recognition, medical image analysis, photo manipulation & target detection.
– Học Biểu diễn Hình ảnh là 1 vấn đề cơ bản trong việc hiểu ngữ nghĩa của nhiều loại dữ liệu hình ảnh, e.g.: ảnh chụp, ảnh y tế, ảnh quét tài liệu, & luồng video. Thông thường, mục tiêu của học biểu diễn hình ảnh trong xử lý hình ảnh là thu hẹp khoảng cách ngữ nghĩa giữa dữ liệu pixel & ngữ nghĩa của hình ảnh. Những thành tựu thành công của học biểu diễn hình ảnh đã giải quyết được nhiều vấn đề thực tế, bao gồm nhưng không giới hạn ở tìm kiếm hình ảnh, nhận dạng khuôn mặt, phân tích hình ảnh y tế, thao tác ảnh & phát hiện mục tiêu.
In recent years, have witnessed a fast advancement of image representation learning from handcrafted feature engineering to that from scratch through deep neural network models. Traditionally, patterns of images are extracted with help of hand-crafted features by human beings based on prior knowledge. E.g., Huang et al (2000) extracted character's structure feature from strokes, then use them to recognize handwritten characters. Rui (2005) adopted morphology method to improve local feature of characters, then use PCA to extract features of characters. However, all of these methods need to extract features from images manually & thus prediction performances strongly rely on prior knowledge. In field of computer vision, manual feature extraction is very cumbersome & impractical because of high dimensionality of feature vectors. Thus, representation learning of images which can automatically extract meaningful, hidden & complex patterns from high-dimension visual data is necessary. DL-based representation learning for images is learned in an end-to-end fashion, which can perform much better than hand-crafted features in target applications, as long as training data is of sufficient quality & quantity.
– Trong những năm gần đây, đã chứng kiến sự tiến bộ nhanh chóng của việc học biểu diễn hình ảnh từ kỹ thuật đặc trưng thủ công sang kỹ thuật từ đầu thông qua các mô hình mạng nơ-ron sâu. Theo truyền thống, các mẫu hình ảnh được trích xuất với sự trợ giúp của các đặc trưng thủ công của con người dựa trên kiến thức trước đó. E.g., Huang & cộng sự (2000) đã trích xuất đặc trưng cấu trúc ký tự từ các nét, sau đó sử dụng chúng để nhận dạng ký tự viết tay. Rui (2005) đã áp dụng phương pháp hình thái để cải thiện đặc trưng cục bộ của các ký tự, sau đó sử dụng PCA để trích xuất các đặc trưng của các ký tự. Tuy nhiên, tất cả các phương pháp này đều cần trích xuất các đặc trưng từ hình ảnh theo cách thủ công & do đó hiệu suất dự đoán phụ thuộc rất nhiều vào kiến thức trước đó. Trong lĩnh vực thị giác máy tính, việc trích xuất đặc trưng thủ công rất cồng kềnh & không thực tế do các vectơ đặc trưng có nhiều chiều. Do đó, việc học biểu diễn hình ảnh có thể tự động trích xuất các mẫu phức tạp, ẩn & có ý nghĩa từ dữ liệu hình ảnh có nhiều chiều là cần thiết. Việc học biểu diễn dựa trên DL cho hình ảnh được học theo cách đầu cuối, có thể hoạt động tốt hơn nhiều so với các tính năng thủ công trong các ứng dụng mục tiêu, miễn là dữ liệu đào tạo có chất lượng & số lượng đủ.
Supervised Representation Learning for image processing. In domain of image processing, supervised learning algorithms, e.g. Convolutional Neural Network (CNN) & Deep Belief Network (DBN), are commonly applied in solving various tasks. 1 of earliest deep-supervised-learning-based works was proposed in 2006 (Hinton et al, 2006), which is focused on MNIST digit image classification problem, outperforming state-of-art SVMs. Following this, deep convolutional neural networks (ConvNets) showed amazing performance which is greatly depends on their properties of shift invariance, weights sharing & local pattern capturing. Different types of network architectures were developed to increase capacity of network models, & larger & larger datasets were collected these days. Various networks including AlexNet (Krizhevsky et al, 2012), VGG (Simonyan & isserman, 2014b), GoogLeNet (Szegedy et al, 2015), ResNet (He et al, 2016a), & DenseNet (Huang et al, 2017a) & large scale datasets, e.g. ImageNet & OpenImage, have been proposed to train very deep convolutional neural networks. With sophisticated architectures & large-scale datasets, performance of convolutional neural networks CNNs keeps outperforming state-of-arts in various computer vision tasks.
– *Học Biểu diễn Có Giám sát cho xử lý hình ảnh.* Trong lĩnh vực xử lý hình ảnh, các thuật toán học có giám sát, e.g. Mạng Nơ-ron Tích chập (CNN) & Mạng Niềm tin Sâu (DBN), thường được áp dụng để giải quyết nhiều tác vụ khác nhau. 1 trong những công trình đầu tiên dựa trên học có giám sát sâu được đề xuất vào năm 2006 (Hinton & cộng sự, 2006), tập trung vào bài toán phân loại ảnh số MNIST, vượt trội hơn các SVM hiện đại. Tiếp theo đó, các mạng nơ-ron tích chập sâu (ConvNet) đã cho thấy hiệu suất đáng kinh ngạc, phụ thuộc rất nhiều vào các đặc tính bất biến dịch chuyển, chia sẻ trọng số & khả năng bắt mẫu cục bộ. Nhiều loại kiến trúc mạng khác nhau đã được phát triển để tăng dung lượng của các mô hình mạng, & các tập dữ liệu lớn hơn & lớn hơn đã được thu thập ngày nay. Nhiều mạng lưới khác nhau, bao gồm AlexNet (Krizhevsky & cộng sự, 2012), VGG (Simonyan & isserman, 2014b), GoogLeNet (Szegedy & cộng sự, 2015), ResNet (He & cộng sự, 2016a), DenseNet (Huang & cộng sự, 2017a) & các tập dữ liệu quy mô lớn, e.g. ImageNet & OpenImage, đã được đề xuất để huấn luyện các mạng nơ-ron tích chập rất sâu. Với kiến trúc phức tạp & tập dữ liệu quy mô lớn, hiệu suất của các mạng nơ-ron tích chập (CNN) vẫn vượt trội hơn các công nghệ tiên tiến trong nhiều tác vụ thị giác máy tính.
Unsupervised Representation Learning for image processing. Collection & annotation of large-scale datasets are time-consuming & expensive in both image datasets & video datasets. E.g., ImageNet contains about 1.3 million labeled images covering 1000 classes while each image is labeled by human workers with 1 class label. To alleviate extensive human annotation labors, many unsupervised methods were proposed to learn visual features from large-scale unlabeled images or videos without using any human annotations. A popular solution: propose various pretext tasks for models to solve, while models can be trained by learning objective functions of pretext tasks & features are learned through this process. Various pretext tasks have been proposed for unsupervised learning, including colorizing gray-scale images (Zhang et al, 2016d) & image inpainting (Pathak et al, 2016). During unsupervised training phase, a predefined pretext tasks is designed for models to solve, & pseudo labels for pretext task are automatically generated based on some attributes to data. Then models are trained according to objective functions of pretext tasks. When trained with pretext tasks, shallower blocks of deep neural network model focus on low-level general features e.g. corners, edges, & textures, while deeper blocks focus on high-level task-specific features e.g. objects, scenes, & object parts. Therefore, models trained with

pretext tasks can learn kernels to capture low-level features & high-level features that are helpful for other downstream tasks. After unsupervised training is finished, learned visual features in this pre-trained models can be further transferred to downstream tasks (especially when only relatively small data is available) to improve performance & overcome overfitting.
– *Học biểu diễn không giám sát cho xử lý hình ảnh.* Thu thập & chú thích các tập dữ liệu quy mô lớn tốn thời gian & tốn kém trong cả tập dữ liệu hình ảnh & tập dữ liệu video. Ví dụ: ImageNet chứa khoảng 1,3 triệu hình ảnh được gắn nhãn bao gồm 1000 lớp trong khi mỗi hình ảnh được dán nhãn bởi công nhân con người với 1 nhãn lớp. Để giảm bớt công việc chú thích của con người, nhiều phương pháp không giám sát đã được đề xuất để học các đặc điểm trực quan từ hình ảnh hoặc video không có nhãn quy mô lớn mà không cần sử dụng bất kỳ chú thích nào của con người. 1 giải pháp phổ biến: đề xuất nhiều tác vụ tiền đề khác nhau để các mô hình giải quyết, trong khi các mô hình có thể được đào tạo bằng cách học các hàm mục tiêu của các tác vụ tiền đề & các đặc điểm được học thông qua quá trình này. Nhiều tác vụ tiền đề khác nhau đã được đề xuất cho học không giám sát, bao gồm tô màu cho hình ảnh thang độ xám (Zhang & cộng sự, 2016d) & tô màu hình ảnh (Pathak & cộng sự, 2016). Trong giai đoạn huấn luyện không giám sát, 1 nhiệm vụ tiền đề được xác định trước được thiết kế để các mô hình giải quyết, & nhãn giả cho nhiệm vụ tiền đề được tự động tạo ra dựa trên 1 số thuộc tính của dữ liệu. Sau đó, các mô hình được huấn luyện theo các hàm mục tiêu của nhiệm vụ tiền đề. Khi được huấn luyện với các nhiệm vụ tiền đề, các khối nông hơn của mô hình mạng nơ-ron sâu tập trung vào các đặc điểm chung cấp thấp, e.g. các góc, cạnh, & kết cấu, trong khi các khối sâu hơn tập trung vào các đặc điểm cụ thể của nhiệm vụ cấp cao, e.g. các đối tượng, cảnh, & các bộ phận của đối tượng. Do đó, các mô hình được huấn luyện với các nhiệm vụ tiền đề có thể học các hạt nhân để nắm bắt các đặc điểm cấp thấp & các đặc điểm cấp cao hữu ích cho các nhiệm vụ hạ lưu khác. Sau khi quá trình huấn luyện không giám sát kết thúc, các đặc điểm trực quan đã học trong các mô hình được huấn luyện trước này có thể được chuyển tiếp sang các nhiệm vụ hạ lưu (đặc biệt là khi chỉ có dữ liệu tương đối nhỏ) để cải thiện hiệu suất & khắc phục tình trạng quá khớp.
Transfer Learning for image processing. In real-world applications, due to high cost of manual labeling, sufficient training data that belongs to same feature space or distribution as testing data may not always be accessible. Transfer learning mimics human vision system by making use of sufficient amounts of prior knowledge in other related domains (i.e., source domains) when executing new tasks in given domain (i.e., target domain). In transfer learning, both training set & test set can contribute to target & source domains. In most cases, there is only 1 target domain for a transfer learning task, while either single or multiple source domains can exist. Techniques of transfer learning in images processing can be categorized into feature representation knowledge transfer & classifier-based knowledge transfer. Specifically, feature representation transfer methods map target domain to source domains by exploiting a set of extracted features, where data divergence between target domain & source domains can be significantly reduced so that performance of task in target domain is improved. E.g., classifier-based knowledge-transfer methods usually share common trait that learned source domain models are utilized as prior knowledge, which are used to learn target model together with training samples. Instead of minimizing cross-domain dissimilarity by updating instances' representations, classifier-based knowledge-transfer methods aim to learn a new model that minimizes generalization error in target domain via provided training set from both domains & learned model.
– *Học chuyển giao cho xử lý hình ảnh.* Trong các ứng dụng thực tế, do chi phí gắn nhãn thủ công cao, dữ liệu đào tạo đủ thuộc cùng không gian đặc trưng hoặc phân phối với dữ liệu kiểm tra có thể không phải lúc nào cũng có thể truy cập được. Học chuyển giao mô phỏng hệ thống thị giác của con người bằng cách sử dụng đủ lượng kiến thức trước đó trong các miền liên quan khác (i.e., miền nguồn) khi thực hiện các tác vụ mới trong miền nhất định (i.e., miền đích). Trong học chuyển giao, cả tập huấn luyện & tập kiểm tra đều có thể đóng góp vào miền đích & miền nguồn. Trong hầu hết các trường hợp, chỉ có 1 miền đích cho 1 tác vụ học chuyển giao, trong khi có thể tồn tại 1 hoặc nhiều miền nguồn. Các kỹ thuật học chuyển giao trong xử lý hình ảnh có thể được phân loại thành chuyển giao kiến thức biểu diễn đặc trưng & chuyển giao kiến thức dựa trên bộ phân loại. Cụ thể, các phương pháp chuyển giao biểu diễn đặc trưng ánh xạ miền đích sang miền nguồn bằng cách khai thác 1 tập hợp các đặc trưng được trích xuất, trong đó độ phân kỳ dữ liệu giữa miền đích & miền nguồn có thể được giảm đáng kể để cải thiện hiệu suất của tác vụ trong miền đích. E.g., các phương pháp chuyển giao tri thức dựa trên bộ phân loại thường có đặc điểm chung là các mô hình miền nguồn đã học được được sử dụng làm tri thức tiên nghiệm, được dùng để học mô hình đích cùng với các mẫu huấn luyện. Thay vì giảm thiểu sự khác biệt giữa các miền bằng cách cập nhật biểu diễn của các thể hiện, các phương pháp chuyển giao tri thức dựa trên bộ phân loại hướng đến việc học 1 mô hình mới giúp giảm thiểu lỗi khái quát hóa trong miền đích thông qua tập huấn luyện được cung cấp từ cả 2 miền & mô hình đã học.
Other Representation Learning for Image Processing. Other types of representation learning are also commonly observed for dealing with image processing, e.g. reinforcement learning, & semi-supervised learning. E.g., reinforcement learning are commonly explored in task of image captioning Liu et al (2018a); Ren et al (2017) & image editing Kosugi & Yamasaki (2020), where learning process is formalized as a sequence of actions based on a policy network.
– *Học Biểu diễn Khác cho Xử lý Hình ảnh.* Các loại học biểu diễn khác cũng thường được quan sát thấy khi xử lý hình ảnh, e.g. học tăng cường, & học bán giám sát. E.g., học tăng cường thường được khám phá trong nhiệm vụ chú thích hình ảnh (Liu & cộng sự (2018a); Ren & cộng sự (2017) & chỉnh sửa hình ảnh) (Kosugi & Yamasaki (2020), trong đó quá trình học được chính thức hóa thành 1 chuỗi hành động dựa trên mạng chính sách).

∗ 1.2.2. Representation Learning for Speech Recognition. Nowadays, speech interfaces or systems have become widely developed & integrated into various real-life applications & devices. Services like Siri [Siri is an AI assistant software built into Apple's iOS system.], Cortana [Microsoft Cortana is an intelligent personal assistant developed by Microsoft, known as "world's 1st cross-platform intelligent personal assistant".], & Google Voice Search [Google Voice Search is a product of Google that allows you to use Google to search by speaking to a mobile phone or computer, i.e., to use legendary content on

device to be identified by server, & then search for information based on results of recognition.] have become a part of our daily lief & are used by millions of users. Exploration in speech recognition & analysis has always been motivated by a desire to enable machines to participate in verbal human-machine interactions. Research goals of enabling machines to understand human speech, identify speakers, & detect human emotion have attracted researchers' attention for > 60 years across several distinct research areas, including but not limited to Automatic Speech Recognition (ASR), Speaker Recognition (SR), & Speaker Emotion Recognition (SER).

– Học biểu diễn cho Nhận dạng giọng nói. Ngày nay, các giao diện hoặc hệ thống giọng nói đã được phát triển rộng rãi & tích hợp vào nhiều ứng dụng & thiết bị thực tế. Các dịch vụ như Siri [Siri là phần mềm trợ lý AI được tích hợp trong hệ thống iOS của Apple.], Cortana [Microsoft Cortana là trợ lý cá nhân thông minh do Microsoft phát triển, được gọi là "trợ lý cá nhân thông minh đa nền tảng đầu tiên trên thế giới".], & Tìm kiếm bằng giọng nói của Google [Tìm kiếm bằng giọng nói của Google là sản phẩm của Google cho phép bạn sử dụng Google để tìm kiếm bằng cách nói chuyện với điện thoại di động hoặc máy tính, i.e., sử dụng nội dung huyền thoại trên thiết bị để được máy chủ nhận dạng, & sau đó tìm kiếm thông tin dựa trên kết quả nhận dạng.] đã trở thành 1 phần trong cuộc sống hàng ngày của chúng ta & được hàng triệu người dùng sử dụng. Khám phá trong nhận dạng giọng nói & phân tích luôn được thúc đẩy bởi mong muốn cho phép máy móc tham gia vào các tương tác bằng lời nói giữa người & máy. Mục tiêu nghiên cứu nhằm giúp máy móc có thể hiểu được giọng nói của con người, nhận dạng người nói & phát hiện cảm xúc của con người đã thu hút sự chú ý của các nhà nghiên cứu trong hơn 60 năm qua trên nhiều lĩnh vực nghiên cứu khác nhau, bao gồm nhưng không giới hạn ở Nhận dạng giọng nói tự động (ASR), Nhận dạng người nói (SR) & Nhận dạng cảm xúc của người nói (SER).

Analyzing & processing speech has been a key application of ML algorithms. Research on speech recognition has traditionally considered task of designing efficient models to accomplish prediction & classification decisions. There are 2 main drawbacks of this approach: 1st, feature engineering is cumbersome & requires human knowledge is introduced above; & 2nd, designed features might not be best for specific speech recognition tasks at hand. This has motivated adoption of recent trends in speech community towards utilization of representation learning techniques, which can learn an intermediate representation of input signal automatically that better fits into task at hand & hence lead to improved performance. Among all these successes, DL-based speech representations play an important role. 1 of major reasons for utilization of representation learning techniques in speech technology: speech data is fundamentally different from 2D image data. Images can be analyzed as a whole or in patches, but speech has to be formatted sequentially to capture temporal dependency & patterns.

– Phân tích & xử lý giọng nói là 1 ứng dụng quan trọng của các thuật toán ML. Nghiên cứu về nhận dạng giọng nói theo truyền thống coi nhiệm vụ thiết kế các mô hình hiệu quả để thực hiện các quyết định dự đoán & phân loại. Có 2 nhược điểm chính của phương pháp này: Thứ nhất, kỹ thuật đặc trưng cồng kềnh & đòi hỏi kiến thức của con người đã được giới thiệu ở trên; Thứ hai, các đặc điểm được thiết kế có thể không phù hợp nhất với các tác vụ nhận dạng giọng nói cụ thể hiện có. Điều này đã thúc đẩy việc áp dụng các xu hướng gần đây trong cộng đồng giọng nói hướng tới việc sử dụng các kỹ thuật học biểu diễn, có thể tự động học 1 biểu diễn trung gian của tín hiệu đầu vào phù hợp hơn với tác vụ hiện có & do đó dẫn đến hiệu suất được cải thiện. Trong số tất cả những thành công này, biểu diễn giọng nói dựa trên DL đóng 1 vai trò quan trọng. 1 trong những lý do chính để sử dụng các kỹ thuật học biểu diễn trong công nghệ giọng nói: dữ liệu giọng nói về cơ bản khác với dữ liệu hình ảnh 2D. Hình ảnh có thể được phân tích toàn bộ hoặc theo từng mảng, nhưng giọng nói phải được định dạng tuần tự để nắm bắt các mẫu & phụ thuộc theo thời gian.

Supervised representation learning for speech recognition. In domain of speech recognition & analyzing, supervised representation learning methods are widely employed, where feature representations are learned on datasets by leveraging label information. E.g., restricted Boltmann machines (RBMs) (Jaitly & Hinton, 2011; Dahl et al, 2010) & deep belief networks (DBNs) (Cairong et al, 2016; Ali et al, 2018) are commonly utilized in learning features from speech for different tasks, including ASR, speaker recognition, & SER. E.g., in 2012, Microsoft has released a new version of their MAVIS (Microsoft Audio Video Indexing Service) speech system based on context-dependent deep neural networks Seide et al, 2011). These authors managed to reduce word error rate on 4 major benchmarks by about 30% (e.g., from 27.4% to 18.5% on RT03S) compared to traditional models based on Gaussian mixtures. Convolutional neural networks are another popular supervised models that are widely utilized for feature learning from speech signals in tasks e.g. speech & speaker recognition (Palaz et al, 2015a,b) & SER Latif et al (2019); Tzirakis et al (2018). Moreover, found: LSTMs (or GRUs) can help CNNs in learning more useful features from speech by learning both local & long-term dependency (Dahl et al, 2010).

– *Học biểu diễn có giám sát cho nhận dạng giọng nói.* Trong lĩnh vực nhận dạng giọng nói & phân tích, các phương pháp học biểu diễn có giám sát được sử dụng rộng rãi, trong đó các biểu diễn đặc trưng được học trên các tập dữ liệu bằng cách tận dụng thông tin nhãn. Ví dụ: máy Boltmann hạn chế (RBM) (Jaitly & Hinton, 2011; Dahl & cộng sự, 2010) & mạng niềm tin sâu (DBN) (Cairong & cộng sự, 2016; Ali & cộng sự, 2018) thường được sử dụng để học các đặc trưng từ giọng nói cho các tác vụ khác nhau, bao gồm ASR, nhận dạng người nói, & SER. Ví dụ: vào năm 2012, Microsoft đã phát hành phiên bản mới của hệ thống giọng nói MAVIS (Dịch vụ Lập chỉ mục Âm thanh & Video của Microsoft) dựa trên mạng nơ-ron sâu phụ thuộc ngữ cảnh (Seide & cộng sự, 2011). Các tác giả này đã giảm tỷ lệ lỗi từ trên 4 phép đo chuẩn chính khoảng 30% (e.g.: từ 27,4% xuống 18,5% trên RT03S) so với các mô hình truyền thống dựa trên hỗn hợp Gauss. Mạng nơ-ron tích chập là 1 mô hình giám sát phổ biến khác được sử dụng rộng rãi để học đặc trưng từ tín hiệu giọng nói trong các tác vụ như nhận dạng giọng nói & người nói (Palaz & cộng sự, 2015a,b) & SER Latif & cộng sự (2019); Tzirakis & cộng sự (2018). Hơn nữa, họ nhận thấy: LSTM (hay GRU) có thể giúp CNN học các đặc trưng hữu ích hơn từ giọng nói bằng cách học cả phụ thuộc cục bộ & dài hạn (Dahl & cộng sự, 2010).

Unsupervised Representation Learning for speech recognition. Unsupervised representation learning from large unlabeled datasets is an active area of speech recognition. In context of speech analysis, able to exploit practically available unlimited

amount of unlabeled corpora to learn good intermediate feature representations, which can then be used to improve performance of a variety of downstream supervised learning speech recognition tasks or speech signal synthetic tasks. In tasks of ASR & SR, most of works are based on Variational Auto-encoder (VAEs), where a generative model & an inference model are jointly learned, which allows them to capture latent representations from observed speech data (Chorowski et al, 2019; Hsu et al, 2019, 2017). E.g., Hsu et al (2017) proposed a hierarchical VAE to capture interpretable & disentangled representations from speech without any supervision. Other auto-encoding architectures like Denoised Autoencoder (DAEs) are also found very promising in finding speech representations in an unsupervised way, especially for noisy speech recognition (Feng et al, 2014; Zhao et al, 2015). Beyond aforementioned, recently, adversarial learning (AL) is emerging as a powerful tool in learning unsupervised representation for speech, e.g. generative adversarial nets (GANs). It involves at least a generator & a discriminator, where former tries to generates as realistic as possible data to obfuscate the latter which also tries its best to deobfuscate. Hence both of generator & discriminator can be trained & improved iteratively in an adversarial way, which result in more discriminative & robust features. Among these, GANs (Chang & Scherer, 2017; Donahue et al, 2018), adversarial autoencoders (AAEs) Sahu et al (2017) are becoming mostly popular in modeling speech not only in ASR but also SR & SER.

– Học biểu diễn không giám sát cho nhận dạng giọng nói. Học biểu diễn không giám sát từ các tập dữ liệu lớn không có nhãn là 1 lĩnh vực năng động của nhận dạng giọng nói. Trong bối cảnh phân tích giọng nói, có thể khai thác số lượng không giới hạn các tập dữ liệu không có nhãn có sẵn trên thực tế để học các biểu diễn đặc trưng trung gian tốt, sau đó có thể được sử dụng để cải thiện hiệu suất của nhiều tác vụ nhận dạng giọng nói học có giám sát hạ lưu hoặc các tác vụ tổng hợp tín hiệu giọng nói. Trong các tác vụ ASR & SR, hầu hết các công trình đều dựa trên Bộ mã hóa tự động biến thiên (VAE), trong đó 1 mô hình sinh & 1 mô hình suy luận được học chung, cho phép chúng nắm bắt các biểu diễn tiềm ẩn từ dữ liệu giọng nói quan sát (Chorowski & cộng sự, 2019; Hsu & cộng sự, 2019, 2017). E.g., Hsu & cộng sự (2017) đã đề xuất 1 VAE phân cấp để nắm bắt các biểu diễn có thể diễn giải & không bị rối từ giọng nói mà không cần bất kỳ sự giám sát nào. Các kiến trúc mã hóa tự động khác như Denoised Autoencoder (DAE) cũng được đánh giá rất hứa hẹn trong việc tìm kiếm biểu diễn giọng nói theo cách không giám sát, đặc biệt là đối với nhận dạng giọng nói có nhiễu (Feng & cộng sự, 2014; Zhao & cộng sự, 2015). Ngoài những kiến trúc đã đề cập ở trên, gần đây, học đối kháng (AL) đang nổi lên như 1 công cụ mạnh mẽ trong việc học biểu diễn giọng nói không giám sát, e.g. mạng đối kháng sinh sinh (GAN). Nó bao gồm ít nhất 1 bộ tạo & 1 bộ phân biệt, trong đó bộ tạo & bộ phân biệt cố gắng tạo ra dữ liệu thực tế nhất có thể để làm tối nghĩa bộ tạo cũng cố gắng hết sức để giải tối nghĩa. Do đó, cả bộ tạo & bộ phân biệt đều có thể được huấn luyện & cải tiến lặp đi lặp lại theo cách đối kháng, dẫn đến các tính năng phân biệt & mạnh mẽ hơn. Trong số đó, GAN (Chang & Scherer, 2017; Donahue & cộng sự, 2018), bộ mã hóa tự động đối nghịch (AAE) Sahu & cộng sự (2017) đang trở nên phổ biến trong việc mô hình hóa giọng nói không chỉ trong ASR mà còn trong SR & SER.

Transfer Learning for speech recognition. Transfer learning (TL) encompasses different approaches, including MTL, model adaptation, knowledge transfer, covariance shift, etc. In domain of speech recognition, representation learning gained much interest in these approaches of TL including but not limited to domain adaptation, multitask learning, & self-taught learning. In terms of Domain Adaptation, speech is a typical example of heterogeneous data & thus, a mismatch always exists between probability distributions of source & target domain area. To build more robust systems for speech-related applications in real-life, domain adaptation techniques are usually applied in training pipeline of deep neural networks to learn representations which are able to explicitly minimize difference between distribution of data in source & target domains (Sun et al, 2017; Swietojanski et al, 2016). In terms of MTL, representations learned can successfully increases performance of speech recognition without requiring contextual speech data, since speech contains multidimensional information (message, speaker, gender, or emotion) that can be used as auxiliary tasks. E.g., in task of ASR, by using MTL with different auxiliary tasks including gender, speaker adaptation, speech enhancement, shown: learned shared representations for different tasks can act as complementary information about acoustics environment & give a lower word error rate (WER) Parthasarathy & Busso, 2017; Xia & Liu, 2015).

– Học chuyển giao cho nhận dạng giọng nói. Học chuyển giao (TL) bao gồm các phương pháp tiếp cận khác nhau, bao gồm MTL, thích ứng mô hình, chuyển giao kiến thức, dịch chuyển hiệp phương sai, v.v. Trong lĩnh vực nhận dạng giọng nói, học biểu diễn đã thu hút được nhiều sự quan tâm trong các phương pháp tiếp cận TL này bao gồm nhưng không giới hạn ở thích ứng miền, học đa nhiệm, & học tự học. Về mặt Thích ứng miền, giọng nói là 1 ví dụ điển hình về dữ liệu không đồng nhất & do đó, luôn tồn tại sự không khớp giữa phân phối xác suất của miền nguồn & miền đích. Để xây dựng các hệ thống mạnh mẽ hơn cho các ứng dụng liên quan đến giọng nói trong đời thực, các kỹ thuật thích ứng miền thường được áp dụng trong đường ống đào tạo của mạng nơ-ron sâu để học các biểu diễn có khả năng giảm thiểu rõ ràng sự khác biệt giữa phân phối dữ liệu trong miền nguồn & miền đích (Sun & cộng sự, 2017; Swietojanski & cộng sự, 2016). Về mặt MTL, các biểu diễn đã học có thể tăng hiệu suất nhận dạng giọng nói 1 cách thành công mà không yêu cầu dữ liệu giọng nói theo ngữ cảnh, vì giọng nói chứa thông tin đa chiều (thông điệp, người nói, giới tính hoặc cảm xúc) có thể được sử dụng làm các tác vụ phụ trợ. E.g., trong nhiệm vụ ASR, bằng cách sử dụng MTL với các nhiệm vụ phụ trợ khác nhau bao gồm giới tính, sự thích ứng của người nói, sự cải thiện giọng nói, đã chỉ ra: các biểu diễn chung đã học được cho các nhiệm vụ khác nhau có thể đóng vai trò là thông tin bổ sung về môi trường âm học & mang lại tỷ lệ lỗi từ thấp hơn (WER) (Parthasarathy & Busso, 2017; Xia & Liu, 2015).

Other Representation Learning for speech recognition. Other than abovementioned 3 categories of representation learning fro speech signals, there are also some other representation learning techniques commonly explored, e.g. semi-supervised learning & reinforcement learning. E.g., in speech recognition for ASR, semi-supervised learning is mainly used to circumvent lack of sufficient training data. This can be achieved either by creating features fronts ends (Thomas et al, 2013), or by using multilingual acoustic representations (Cui et al, 2015), or by extracting an intermediate representation from large unpaired datasets (Karita et al, 2018). RL is also gaining interest in area of speech recognition, & there have been

multiple approaches to model different speech problems, including dialog modeling & optimization (Levin et al, 2000), speech recognition (Shen et al, 2019), & emotion recognition (Sangeetha & Jayasankar, 2019).

– Học biểu diễn khác cho nhận dạng giọng nói. Ngoài 3 loại học biểu diễn cho tín hiệu giọng nói đã đề cập ở trên, còn có 1 số kỹ thuật học biểu diễn khác thường được khám phá, e.g. học bán giám sát & học tăng cường. E.g., trong nhận dạng giọng nói cho ASR, học bán giám sát chủ yếu được sử dụng để khắc phục tình trạng thiếu dữ liệu huấn luyện. Điều này có thể đạt được bằng cách tạo các đặc trưng ở đầu cuối (Thomas & cộng sự, 2013), hoặc bằng cách sử dụng biểu diễn âm thanh đa ngôn ngữ (Cui & cộng sự, 2015), hoặc bằng cách trích xuất biểu diễn trung gian từ các tập dữ liệu lớn không ghép đôi (Karita & cộng sự, 2018). RL cũng đang thu hút sự quan tâm trong lĩnh vực nhận dạng giọng nói, & đã có nhiều phương pháp để mô hình hóa các vấn đề giọng nói khác nhau, bao gồm mô hình hóa hội thoại & tối ưu hóa (Levin & cộng sự, 2000), nhận dạng giọng nói (Shen & cộng sự, 2019), & nhận dạng cảm xúc (Sangeetha & Jayasankar, 2019).

* 1.2.3. Representation Learning for Natural Language Processing. p. 10+++
* 1.2.4. Representation Learning for Networks. p. 13+++
* Summary. Representation learning is a very active & important field currently, which heavily influences effectiveness of ML techniques. Representation learning is about learning representations of data that makes it easier to extract useful & discriminative information when building classifiers or other predictors. Among various ways of learning representations, DL algorithms have increasingly been employed in many areas nowadays where good representation can be learned in an efficient & automatic way based on large amount of complex & high dimensional data. Evaluation of a representation is closely related to its performance on downstream tasks. Generally, there are also some general properties that good representations may hold, e.g. smoothness, linearity, disentanglement, as well as capturing multiple explanatory & casual factors.

  – Học biểu diễn là 1 lĩnh vực rất năng động & quan trọng hiện nay, ảnh hưởng lớn đến hiệu quả của các kỹ thuật ML. Học biểu diễn liên quan đến việc học các biểu diễn dữ liệu giúp trích xuất thông tin & phân biệt hữu ích dễ dàng hơn khi xây dựng bộ phân loại hoặc các yếu tố dự đoán khác. Trong số nhiều cách học biểu diễn, các thuật toán DL ngày càng được sử dụng rộng rãi trong nhiều lĩnh vực, nơi biểu diễn tốt có thể được học 1 cách hiệu quả & tự động dựa trên lượng lớn dữ liệu phức tạp & đa chiều. Đánh giá 1 biểu diễn có liên quan chặt chẽ đến hiệu suất của nó trong các tác vụ hạ nguồn. Nhìn chung, cũng có 1 số đặc tính chung mà các biểu diễn tốt có thể sở hữu, e.g. độ mượt, tính tuyến tính, độ tách rời, cũng như nắm bắt được nhiều yếu tố giải thích & ngẫu nhiên.

  Summarized representation learning techniques in different domains, focusing on unique challenges & models for different areas including processing of images, natural language, & speech signals. For each area, there emerges many DL-based representation techniques from different categories, including supervised learning, unsupervised learning, transfer learning, disentangled representation learning, reinforcement learning, etc. Have also briefly mentioned about representation learning on networks & its relations to that on images, texts, & speech, in order for elaboration of it in following chaps.

  – Tóm tắt các kỹ thuật học biểu diễn trong các lĩnh vực khác nhau, tập trung vào những thách thức & mô hình độc đáo cho các lĩnh vực khác nhau, bao gồm xử lý hình ảnh, ngôn ngữ tự nhiên, & tín hiệu giọng nói. Đối với mỗi lĩnh vực, có nhiều kỹ thuật biểu diễn dựa trên DL từ các danh mục khác nhau, bao gồm học có giám sát, học không giám sát, học chuyển giao, học biểu diễn không rối rắm, học tăng cường, v.v. Bài viết cũng đề cập ngắn gọn về học biểu diễn trên mạng & mối quan hệ của nó với hình ảnh, văn bản, & giọng nói, để có thể trình bày chi tiết hơn trong các chương sau.

- 2. Peng Cui, Lingfei Wu, Jian Pei, Liang Zhao, Xiao Wang. Graph Representation Learning. Graph representation learning aims at assigning nodes in a graph to low-dimensional representations & effectively preserving graph structures. Recently, a significant amount of progress has been made toward this emerging graph analysis paradigm. In this chap, 1st summarize motivation of graph representation learning. Afterwards & primarily, provide a comprehensive overview of a large number of graph representation learning methods in a systematic manner, covering traditional graph representation learning, modern graph representation learning, & GNNs.

  – Học biểu diễn đồ thị hướng đến việc gán các nút trong đồ thị cho các biểu diễn số chiều thấp & bảo toàn hiệu quả cấu trúc đồ thị. Gần đây, mô hình phân tích đồ thị mới nổi này đã đạt được nhiều tiến bộ đáng kể. Trong chương này, trước tiên, chúng tôi tóm tắt động lực của việc học biểu diễn đồ thị. Sau đó, chúng tôi chủ yếu cung cấp tổng quan toàn diện về 1 số lượng lớn các phương pháp học biểu diễn đồ thị 1 cách có hệ thống, bao gồm học biểu diễn đồ thị truyền thống, học biểu diễn đồ thị hiện đại, & Mạng nơ-ron nhân tạo (GNN).

  ◦ 2.1. Graph Representation Learning: An Introduction. Many complex systems take form of graphs, e.g. social networks, biological networks, & information networks. Well recognized: graph data is often sophisticated & thus is challenging to deal with. To process graph data effectively, 1st critical challenge: find effective graph data representation, i..e, how to represent graphs concisely so that advanced analytic tasks, e.g. pattern discovery, analysis, & prediction, can be conducted efficiently in both time & space. Traditionally, usually represent a graph as $G = (V, E)$, where V is a node set & E is an edge set. For large graphs, e.g. those with billions of nodes, traditional graph representation poses several challenges to graph processing & analysis.

    – Học Biểu diễn Đồ thị: Giới thiệu. Nhiều hệ thống phức tạp có dạng đồ thị, e.g.: mạng xã hội, mạng sinh học, & mạng thông tin. Nhận thức rõ ràng: dữ liệu đồ thị thường phức tạp & do đó rất khó xử lý. Để xử lý dữ liệu đồ thị hiệu quả, thách thức quan trọng đầu tiên là: tìm cách biểu diễn dữ liệu đồ thị hiệu quả, i.e., cách biểu diễn đồ thị 1 cách ngắn gọn để các tác vụ phân tích nâng cao, e.g.: khám phá mẫu, phân tích, & dự đoán, có thể được thực hiện hiệu quả trong cả không gian & thời gian. Theo truyền thống, biểu diễn đồ thị thường là $G = (V, E)$, trong đó V là tập nút & E là tập cạnh. Đối với các đồ thị lớn, e.g.: đồ thị có hàng tỷ nút, biểu diễn đồ thị truyền thống đặt ra 1 số thách thức cho việc xử lý & phân tích đồ thị.

1. **High computational complexity.** These relationships encoded by edge set E take most of graph processing or analysis algorithms either iterative or combinatorial computation steps. E.g., a popular way: use shortest or average path length between 2 nodes to represent their distance. To compute such a distance using traditional graph representation, have to enumerate many possible paths between 2 nodes, which is in nature a combinatorial problem. Such methods result in high computational complexity that prevents them from being applicable to large-scale real-world graphs.
   – **Độ phức tạp tính toán cao.** Các mối quan hệ được mã hóa bởi tập cạnh E này đòi hỏi hầu hết các thuật toán xử lý hoặc phân tích đồ thị phải thực hiện các bước tính toán lặp hoặc kết hợp. E.g., 1 cách phổ biến: sử dụng độ dài đường đi ngắn nhất hoặc trung bình giữa 2 nút để biểu diễn khoảng cách của chúng. Để tính khoảng cách như vậy bằng cách sử dụng biểu diễn đồ thị truyền thống, cần phải liệt kê nhiều đường đi khả thi giữa 2 nút, về bản chất là 1 bài toán kết hợp. Các phương pháp như vậy dẫn đến độ phức tạp tính toán cao, khiến chúng không thể áp dụng cho các đồ thị thực tế quy mô lớn.
2. **Low parallelizability.** Parallel & distributed computing is de facto to processes & analyze large-scale data. Graph data represented in traditional way, however, casts severe difficulties to design & implement of parallel & distributed algorithms. Bottleneck: nodes in a graph are coupled to each other explicitly reflected by E. Thus, distributing different nodes in different shards or servers often causes demandingly high communication cost among servers, & holds back speed-up ratio.
   – **Khả năng song song hóa thấp.** Tính toán song song & phân tán thực tế là để xử lý & phân tích dữ liệu quy mô lớn. Tuy nhiên, dữ liệu đồ thị được biểu diễn theo cách truyền thống gây ra những khó khăn nghiêm trọng trong việc thiết kế & triển khai các thuật toán song song & phân tán. Nút thắt cổ chai: các nút trong đồ thị được kết nối với nhau 1 cách rõ ràng, được phản ánh bởi E. Do đó, việc phân phối các nút khác nhau trong các phân đoạn hoặc máy chủ khác nhau thường gây ra chi phí truyền thông cao giữa các máy chủ, & làm giảm tốc độ xử lý.
3. **Inapplicability of ML methods.** Recently, ML methods, especially DL, are very powerful in many areas. For graph data represented in traditional way, however, most of off-shelf ML methods may not be applicable. Those methods usually assume: data samples can be represented by independent vectors in a vector space, while samples in graph data (i.e., nodes) are dependent to each other to some degree determined by E. Although can simply represent a node by its corresponding row vector in adjacency matrix of graph, extremely high dimensionality of such a representation in a large graph with many nodes makes the in sequel graph processing & analysis difficult.
   – **Tính không áp dụng được của các phương pháp ML.** Gần đây, các phương pháp ML, đặc biệt là DL, rất mạnh mẽ trong nhiều lĩnh vực. Tuy nhiên, đối với dữ liệu đồ thị được biểu diễn theo cách truyền thống, hầu hết các phương pháp ML có sẵn có thể không áp dụng được. Các phương pháp này thường giả định: các mẫu dữ liệu có thể được biểu diễn bằng các vectơ độc lập trong không gian vectơ, trong khi các mẫu trong dữ liệu đồ thị (i.e., các nút) phụ thuộc lẫn nhau ở 1 mức độ nào đó được xác định bởi E. Mặc dù có thể biểu diễn 1 nút đơn giản bằng vectơ hàng tương ứng của nó trong ma trận kề của đồ thị, nhưng tính đa chiều cực cao của biểu diễn như vậy trong 1 đồ thị lớn với nhiều nút khiến việc xử lý & phân tích đồ thị tiếp theo trở nên khó khăn.

To tackle these challenges, substantial effort has been committed to develop novel graph representation learning, i.e., learning dense & continuous low-dimensional vector representations for nodes, so that noise or redundant information can be reduced & intrinsic structure information can be preserved. In learned representation space, relationships among nodes, which were originally represented by edges or other high-order topological measures in graphs, are captures by distances between nodes in vector space, & structural characteristics of a node are encoded into its representation vector.

– Để giải quyết những thách thức này, nhiều nỗ lực đáng kể đã được thực hiện để phát triển phương pháp học biểu diễn đồ thị mới, i.e., học các biểu diễn vectơ liên tục, chiều thấp & dày đặc cho các nút, nhờ đó có thể giảm nhiễu hoặc thông tin dư thừa & bảo toàn thông tin cấu trúc nội tại. Trong không gian biểu diễn đã học, các mối quan hệ giữa các nút, ban đầu được biểu diễn bằng các cạnh hoặc các phép đo tôpô bậc cao khác trong đồ thị, được nắm bắt bằng khoảng cách giữa các nút trong không gian vectơ, & các đặc điểm cấu trúc của 1 nút được mã hóa vào vectơ biểu diễn của nó.

Basically, in order to make representation space well supporting graph analysis tasks, there are 2 goals for graph representation learning. 1st, original graph can be reconstructed from learned representation space. It requires: if there is an edge or relationship between 2 nodes, then distance of these 2 nodes in representation space should be relatively small. 2nd, learned representation space can effectively support graph inference, e.g. predicting unseen links, identifying important nodes, & inferring node labels. It should ne noted: a representation space with only goal of graph reconstruction is not sufficient for graph inference. After representation is obtained, downstream tasks e.g. node classification, node clustering, graph visualization & link prediction can be dealt with based on these representations. Overall, there are 3 main categories of graph representation learning methods: traditional graph embedding, modern graph embedding, & GNNs.

– Về cơ bản, để tạo không gian biểu diễn hỗ trợ tốt cho các tác vụ phân tích đồ thị, có 2 mục tiêu cho việc học biểu diễn đồ thị. Thứ nhất, đồ thị gốc có thể được tái tạo từ không gian biểu diễn đã học. Điều này yêu cầu: nếu có 1 cạnh hoặc mối quan hệ giữa 2 nút, thì khoảng cách của 2 nút này trong không gian biểu diễn phải tương đối nhỏ. Thứ hai, không gian biểu diễn đã học có thể hỗ trợ hiệu quả cho suy luận đồ thị, e.g.: dự đoán các liên kết chưa thấy, xác định các nút quan trọng, & suy ra nhãn nút. Cần lưu ý: không gian biểu diễn chỉ có mục tiêu tái tạo đồ thị là không đủ cho suy luận đồ thị. Sau khi có được biểu diễn, các tác vụ hạ nguồn e.g.: phân loại nút, nhóm nút, trực quan hóa đồ thị & dự đoán liên kết có thể được xử lý dựa trên các biểu diễn này. Nhìn chung, có 3 loại chính của các phương pháp học biểu diễn đồ thị: nhúng đồ thị truyền thống, nhúng đồ thị hiện đại, & GNN.

- 2.2. Traditional Graph Embedding. Traditional graph embedding methods are originally studied as dimension reduction techniques. A graph is usually constructed from a feature represented data set, like image data set. Graph embedding usually

has 2 goals, i.e., reconstructing original graph structures & support graph inference. Objective functions of traditional graph embedding methods mainly target goal of graph reconstruction.

– Nhúng Đồ Thị Truyền Thống. Các phương pháp nhúng đồ thị truyền thống ban đầu được nghiên cứu như các kỹ thuật giảm chiều. 1 đồ thị thường được xây dựng từ 1 tập dữ liệu được biểu diễn bằng đặc trưng, chẳng hạn như tập dữ liệu ảnh. Nhúng đồ thị thường có 2 mục tiêu: tái tạo cấu trúc đồ thị gốc & hỗ trợ suy luận đồ thị. Hàm mục tiêu của các phương pháp nhúng đồ thị truyền thống chủ yếu hướng đến mục tiêu tái tạo đồ thị.

Specifically, Tenenbaum et al (2000) 1st constructs a neighborhood graph G using connectivity algorithms e.g. K nearest neighbors (KNN). Then based on G, shortest path between different data can be computed. Consequently, for all N data entries in data set, have matrix of graph distances. Finally, classical multidimensional scaling (MDS) method is applied to matrix to obtain coordinate vectors. Representations learned by Isomap approximately preserve geodesic distances of entry pairs in low-dimensional space. Key problem of Isomap is its high complexity due to computing of pairwise shortest paths. Locally linear embedding (LLE) (Roweis & Saul, 2000) is proposed to eliminate need to estimate pairwise distances between widely separated entries. LLE assumes: each entry & its neighbors lie on or close to a locally linear patch of a manifold. To characterize local geometry, each entry can be reconstructed from its neighbors. Finally, in low-dimensional space, LLE construct a neighborhood-preserving mapping based on locally linear reconstruction Laplacian eigenmaps (LE) (Belkin & Niyogi, 2002) also begins with constructing a graph using ε-neighborhoods or K nearest neighbors. Then heat kernel (Berline et al, 2003) is utilized to choose weight of 2 nodes in graph. Finally, node representations can be obtained by based on Laplacian matrix regularization. Furthermore, locality preserving projection (LPP) (Berline et al, 2003), a linear approximation of nonlinear LE, is proposed.

– Cụ thể, Tenenbaum & cộng sự (2000) lần đầu tiên xây dựng 1 đồ thị lân cận G bằng các thuật toán kết nối, e.g. K láng giềng gần nhất (KNN). Sau đó, dựa trên G, đường đi ngắn nhất giữa các dữ liệu khác nhau có thể được tính toán. Do đó, với mọi N mục dữ liệu trong tập dữ liệu, có ma trận khoảng cách đồ thị. Cuối cùng, phương pháp tỷ lệ đa chiều cổ điển (MDS) được áp dụng cho ma trận để thu được các vectơ tọa độ. Các biểu diễn được học bởi Isomap gần đúng bảo toàn khoảng cách trắc địa của các cặp mục trong không gian chiều thấp. Vấn đề chính của Isomap là độ phức tạp cao do tính toán các đường đi ngắn nhất từng cặp. Nhúng tuyến tính cục bộ (LLE) (Roweis & Saul, 2000) được đề xuất để loại bỏ nhu cầu ước tính khoảng cách từng cặp giữa các mục cách xa nhau. LLE giả định: mỗi mục & các hàng xóm của nó nằm trên hoặc gần 1 bản vá tuyến tính cục bộ của 1 đa tạp. Để mô tả hình học cục bộ, mỗi mục có thể được tái tạo từ các hàng xóm của nó. Cuối cùng, trong không gian ít chiều, LLE xây dựng 1 ánh xạ bảo toàn lân cận dựa trên phép tái tạo tuyến tính cục bộ các phép riêng Laplacian (LE) (Belkin & Niyogi, 2002) cũng bắt đầu bằng việc xây dựng 1 đồ thị sử dụng các lân cận ε hoặc K lân cận gần nhất. Sau đó, hạt nhân nhiệt (Berline & cộng sự, 2003) được sử dụng để chọn trọng số của 2 nút trong đồ thị. Cuối cùng, các biểu diễn nút có thể thu được bằng cách dựa trên phép chính quy hóa ma trận Laplacian. Hơn nữa, phép chiếu bảo toàn lân cận (LPP) (Berline & cộng sự, 2003), 1 phép xấp xỉ tuyến tính của LE phi tuyến tính, đã được đề xuất.

These methods are extended in rich literature of graph embedding by considering different characteristics of constructed graphs (Fu & Ma, 2012). Can find: traditional graph embedding mostly works on graphs constructed from feature represented data sets, where proximity among nodes encoded by edge weights is well defined in original feature space. While, in contrast, modern graph embedding mostly works on naturally formed networks, e.g. social networks, biology networks, & e-commerce networks. In those networks, proximities among nodes are not explicitly or directly defined. E.g., an edge between 2 nodes usually just implies there is a relationship between them, but cannot indicate specific proximity. Also, even if there is no edge between 2 nodes, cannot say proximity between these 2 nodes is 0. Def of node proximities depends on specific analytic tasks & application scenarios. Therefore, modern graph embedding usually incorporates rich information, e.g. network structures, properties, side information & advanced information, to facilitate different problems & applications. Modern graph embedding needs to target both of goals mentioned before. In view of this, traditional graph embedding can be regarded as a special case of modern graph embedding, & recent research progress on modern graph embedding pays more attention to network inference.

– Các phương pháp này được mở rộng trong các tài liệu phong phú về nhúng đồ thị bằng cách xem xét các đặc điểm khác nhau của đồ thị được xây dựng (Fu & Ma, 2012). Có thể thấy: nhúng đồ thị truyền thống chủ yếu hoạt động trên các đồ thị được xây dựng từ các tập dữ liệu được biểu diễn bằng đặc trưng, trong đó độ gần giữa các nút được mã hóa bởi trọng số cạnh được xác định rõ trong không gian đặc trưng gốc. Ngược lại, nhúng đồ thị hiện đại chủ yếu hoạt động trên các mạng được hình thành tự nhiên, e.g.: mạng xã hội, mạng sinh học, & mạng thương mại điện tử. Trong các mạng đó, độ gần giữa các nút không được xác định rõ ràng hoặc trực tiếp. Ví dụ: 1 cạnh giữa 2 nút thường chỉ ngụ ý có mối quan hệ giữa chúng, nhưng không thể chỉ ra độ gần cụ thể. Ngoài ra, ngay cả khi không có cạnh giữa 2 nút, cũng không thể nói rằng độ gần giữa 2 nút này bằng 0. Định nghĩa về độ gần của các nút phụ thuộc vào các tác vụ phân tích & kịch bản ứng dụng cụ thể. Do đó, nhúng đồ thị hiện đại thường kết hợp thông tin phong phú, e.g.: cấu trúc mạng, thuộc tính, thông tin phụ & thông tin nâng cao, để tạo điều kiện cho các vấn đề & ứng dụng khác nhau. Nhúng đồ thị hiện đại cần hướng đến cả 2 mục tiêu đã đề cập trước đó. Theo quan điểm này, nhúng đồ thị truyền thống có thể được coi là 1 trường hợp đặc biệt của nhúng đồ thị hiện đại, & tiến bộ nghiên cứu gần đây về nhúng đồ thị hiện đại chú trọng hơn đến suy luận mạng.

- 2.3. Modern Graph Embedding. To well support network inference, modern graph embedding considers much richer information in a graph. According to types of information that are preserved in graph representation learning, existing methods can be categorized into 3 categories:

  1. graph structures & properties preserving graph embedding,
  2. graph representation learning with side information, &

3. advanced information preserving graph representation learning.

In technique view, different models are adopted to incorporate different types of information or address different goals. Commonly used models include matrix factorization, random walk, deep neural networks & their variations.

– 2.3. Nhúng Đồ thị Hiện đại. Để hỗ trợ tốt cho suy luận mạng, nhúng đồ thị hiện đại xem xét thông tin phong phú hơn nhiều trong đồ thị. Theo các loại thông tin được bảo toàn trong học biểu diễn đồ thị, các phương pháp hiện có có thể được phân loại thành 3 loại:

1. cấu trúc đồ thị & thuộc tính bảo toàn nhúng đồ thị,
2. học biểu diễn đồ thị với thông tin phụ, &
3. học biểu diễn đồ thị bảo toàn thông tin nâng cao.

Về mặt kỹ thuật, các mô hình khác nhau được áp dụng để kết hợp các loại thông tin khác nhau hoặc giải quyết các mục tiêu khác nhau. Các mô hình thường được sử dụng bao gồm phân tích ma trận, bước ngẫu nhiên, mạng nơ-ron sâu & các biến thể của chúng.

* 2.3.1. Structure-Property Preserving Graph Representation Learning.
  · 2.3.1.1. Structure Preserving Graph Representation Learning.
  · 2.3.1.2. Property Preserving Graph Representation Learning.
* 2.3.2. Graph Representation Learning with Side Information.
* 2.3.3. Advanced Information Preserving Graph Representation Learning.

◦ 2.4. GNNs. Over past decade, DL has become "crown jewel" of AI & ML, showing superior performance in acoustics, images & NLP, etc. Although well known: graphs are ubiquitous in real world, very challenging to utilize DL methods to analyze graph data. This problem is nontrivial because of following challenges:

1. Irregular structures of graphs. Unlike images, audio, & text, which have a clear grid structure, graphs have irregular structures, making it hard to generalize some of basic mathematical operations to graphs. E.g., defining convolution & pooling operations, which are fundamental operations in convolutional neural networks (CNNs), for graph data is not straightforward.
2. Heterogeneity & diversity of graphs. A graph itself can be complicated, containing diverse types & properties. These diverse types, properties, & tasks require different model architectures to tackle specific problems.
3. Large-scale graphs. In big-data era, real graphs can easily have millions or billions of nodes & edges. How to design scalable models, preferably models that have a linear time complexity w.r.t. graph size, is a key problem.
4. Incorporating interdisciplinary knowledge. Graphs are often connected to other disciplines, e.g. biology, chemistry, & social sciences. This interdisciplinary nature provides both opportunities & challenges: domain knowledge can be leveraged to solve specific problems but integrating domain knowledge can complicate model designs.

– Trong thập kỷ qua, DL đã trở thành "viên ngọc quý" của AI & ML, thể hiện hiệu suất vượt trội trong âm học, hình ảnh & NLP, v.v. Mặc dù đã được biết đến rộng rãi: đồ thị có mặt khắp nơi trong thế giới thực, nhưng việc sử dụng các phương pháp DL để phân tích dữ liệu đồ thị lại rất khó khăn. Vấn đề này không hề đơn giản vì những thách thức sau:

1. Cấu trúc đồ thị bất quy tắc. Không giống như hình ảnh, âm thanh, & văn bản có cấu trúc lưới rõ ràng, đồ thị có cấu trúc bất quy tắc, khiến việc khái quát hóa 1 số phép toán cơ bản thành đồ thị trở nên khó khăn. Ví dụ: việc xác định các phép toán tích chập & gộp, vốn là các phép toán cơ bản trong mạng nơ-ron tích chập (CNN), cho dữ liệu đồ thị không hề đơn giản.
2. Tính không đồng nhất & đa dạng của đồ thị. Bản thân 1 đồ thị có thể phức tạp, chứa đựng nhiều loại & thuộc tính khác nhau. Những loại, thuộc tính, & tác vụ đa dạng này đòi hỏi các kiến trúc mô hình khác nhau để giải quyết các vấn đề cụ thể.
3. Đồ thị quy mô lớn. Trong kỷ nguyên dữ liệu lớn, đồ thị thực tế có thể dễ dàng có hàng triệu hoặc hàng tỷ nút & cạnh. Làm thế nào để thiết kế các mô hình có khả năng mở rộng, tốt nhất là các mô hình có độ phức tạp thời gian tuyến tính so với kích thước đồ thị, là 1 vấn đề then chốt.
4. Kết hợp kiến thức liên ngành. Đồ thị thường được kết nối với các ngành khác, e.g.: sinh học, hóa học, & khoa học xã hội. Bản chất liên ngành này mang lại cả cơ hội & thách thức: kiến thức chuyên ngành có thể được tận dụng để giải quyết các vấn đề cụ thể, nhưng việc tích hợp kiến thức chuyên ngành có thể làm phức tạp thiết kế mô hình.

Currently, GNNs have attracted considerable research attention over past several years. Adopted architectures & training strategies vary greatly, ranging from supervised to unsupervised & from convolution to recursive, including graph recurrent neural networks (Graph RNNs) , graph convolutional networks (GCNs), graph autoencoders (GAEs), graph reinforcement learning (Graph GL), & graph adversarial methods. Specifically, Graroperty h RNNs capture recursive & sequential patterns of graphs by modeling states at either node-level or graph-level; GCNs define convolution & readout operations on irregular graph structures to capture common local & global structural patterns; GAEs assume low-rank graph structures & adopt unsupervised methods for node representation learning; Graph RL defines graph-based actions & rewards to obtain feedbacks on graph tasks while following constraints; Graph adversarial methods adopt adversarial training techniques to enhance generalization ability of graphbased models & test their robustness by adversarial attacks.

– Hiện nay, GNN đã thu hút được sự chú ý nghiên cứu đáng kể trong vài năm qua. Các kiến trúc được áp dụng & các chiến lược đào tạo rất khác nhau, từ có giám sát đến không giám sát & từ tích chập đến đệ quy, bao gồm mạng nơ-ron hồi quy đồ thị (Graph RNN), mạng tích chập đồ thị (GCN), bộ mã hóa tự động đồ thị (GAE), học tăng cường đồ thị (Graph GL)

& các phương pháp đối kháng đồ thị. Cụ thể, Graroperty h RNN nắm bắt các mẫu đệ quy & tuần tự của đồ thị bằng cách mô hình hóa các trạng thái ở cấp độ nút hoặc cấp độ đồ thị; GCN định nghĩa tích chập & các phép toán đọc trên các cấu trúc đồ thị bất thường để nắm bắt các mẫu cấu trúc cục bộ & toàn cục chung; GAE giả định các cấu trúc đồ thị hạng thấp & áp dụng các phương pháp không giám sát để học biểu diễn nút; Graph RL định nghĩa các hành động dựa trên đồ thị & phần thưởng để có được phản hồi về các tác vụ đồ thị trong khi tuân theo các ràng buộc; Các phương pháp đối kháng đồ thị áp dụng các kỹ thuật đào tạo đối kháng để tăng cường khả năng khái quát hóa của các mô hình dựa trên đồ thị & kiểm tra tính mạnh mẽ của chúng bằng các cuộc tấn công đối kháng.

There are many ongoing or future research directions which are also worthy of further study, including new models for unstudied graph structures, compositionality of existing models, dynamic graphs, interpretability & robustness, etc. On whole, DL on graphs is a promising & fast-developing research field that both offers exciting opportunities & presents many challenges. Studying DL on graphs constitutes a critical building block in modeling relational data, & it is an important step towards a future with better ML & AI techniques.

– Có rất nhiều hướng nghiên cứu đang & sẽ được triển khai cũng đáng được nghiên cứu sâu hơn, bao gồm các mô hình mới cho cấu trúc đồ thị chưa được nghiên cứu, tính tổng hợp của các mô hình hiện có, đồ thị động, khả năng diễn giải & tính mạnh mẽ, v.v. Nhìn chung, DL trên đồ thị là 1 lĩnh vực nghiên cứu đầy hứa hẹn & đang phát triển nhanh chóng, vừa mang đến những cơ hội thú vị & vừa đặt ra nhiều thách thức. Nghiên cứu DL trên đồ thị là 1 nền tảng quan trọng trong việc mô hình hóa dữ liệu quan hệ, & đây là 1 bước tiến quan trọng hướng tới tương lai với các kỹ thuật ML & AI tốt hơn.

- 2.5. Summary. In this chap, introduce motivation of graph representation learning. Then in Sect. 2, discuss traditional graph embedding methods & modern graph embedding methods are introduced in Sect. 3. Basically, structure & property preserving graph representation learning is foundation. If one cannot preserve well graph structures & retain important graph properties in representation space, serious information will be lost, which hurts analytic tasks in sequel. Based on structures & property preserving graph representation learning, one may apply off-shelf ML methods. If some side information is available, it can be incorporated into graph representation learning. Furthermore, domain knowledge of some certain applications as advanced information can be considered. As shown in Sect. 4, utilizing DL methods on graphs is a promising & fast-developing research field that both offers exciting opportunities & presents many challenges. Studying DL on graphs constitutes a critical building block in modeling relational data, & it is an important step toward a future with better ML & AI techniques.

  – Trong chương này, giới thiệu động lực của việc học biểu diễn đồ thị. Sau đó, trong Phần 2, thảo luận về các phương pháp nhúng đồ thị truyền thống & các phương pháp nhúng đồ thị hiện đại được giới thiệu trong Phần 3. Về cơ bản, học biểu diễn đồ thị bảo toàn cấu trúc & tính chất là nền tảng. Nếu không thể bảo toàn tốt các cấu trúc đồ thị & giữ lại các tính chất quan trọng của đồ thị trong không gian biểu diễn, thông tin quan trọng sẽ bị mất, gây ảnh hưởng đến các tác vụ phân tích sau này. Dựa trên học biểu diễn đồ thị bảo toàn cấu trúc & tính chất, người ta có thể áp dụng các phương pháp ML có sẵn. Nếu có sẵn 1 số thông tin phụ, nó có thể được kết hợp vào học biểu diễn đồ thị. Hơn nữa, kiến thức chuyên môn về 1 số ứng dụng nhất định dưới dạng thông tin nâng cao có thể được coi là thông tin nâng cao. Như đã trình bày trong Phần 4, việc sử dụng các phương pháp DL trên đồ thị là 1 lĩnh vực nghiên cứu đầy hứa hẹn & phát triển nhanh chóng, vừa mang đến những cơ hội thú vị & đặt ra nhiều thách thức. Nghiên cứu DL trên đồ thị là 1 nền tảng quan trọng trong việc mô hình hóa dữ liệu quan hệ, & đây là 1 bước quan trọng hướng tới tương lai với các kỹ thuật ML & AI tốt hơn.

- 3. Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, Le Song. Graph Neural Networks.

  PART II. FOUNDATIONS OF GRAPH NEURAL NETWORKS.

- 4. Jian Tang, Renjie Liao. Graph Neural Networks for Node Classification. GNNs are neural architectures specifically designed for graph-structured data, which have been receiving increasing attention recently & applied to different domains & applications. In this chap, focus on a fundamental task on graphs: node classification. Give a detailed definition of node classification & also introduce some classical approaches e.g. label propagation. Afterwards, introduce a few representative architectures of GNNs for node classification. Further point out main difficulty – oversmoothing problem – of training deep graph neural networks & present some latest advancement along this direction e.g. continuous GNNs.

  – GNN là kiến trúc nơ-ron được thiết kế riêng cho dữ liệu có cấu trúc đồ thị, gần đây đang ngày càng được quan tâm & ứng dụng trong nhiều lĩnh vực & ứng dụng khác nhau. Trong chương này, chúng tôi tập trung vào 1 nhiệm vụ cơ bản trên đồ thị: phân loại nút. Đưa ra định nghĩa chi tiết về phân loại nút & đồng thời giới thiệu 1 số phương pháp tiếp cận cổ điển, e.g. lan truyền nhãn. Sau đó, chúng tôi giới thiệu 1 số kiến trúc GNN tiêu biểu cho phân loại nút. Chúng tôi cũng chỉ ra khó khăn chính – vấn đề làm mịn quá mức – của việc huấn luyện mạng nơ-ron đồ thị sâu & trình bày 1 số tiến bộ mới nhất theo hướng này, e.g. GNN liên tục.

  - 4.1. Background & Problem Def. Graph-structured data (e.g., social networks, WWW, protein-protein interaction networks) are ubiquitous in real-world, covering a variety of applications. A fundamental task on graphs is node classification, which tries to classify nodes into a few predefined categories. E.g., in social networks, want to predict political bias of each user; in protein-protein interaction networks, interested in predicting function role of each protein; in WWW, may have to classify web pages into different semantic categories. To make effective prediction, a critical problem is to have very effective node representations, which largely determine performance of node classification.

    – Dữ liệu có cấu trúc đồ thị (e.g.: mạng xã hội, WWW, mạng tương tác protein-protein) rất phổ biến trong thế giới thực, bao gồm nhiều ứng dụng khác nhau. 1 nhiệm vụ cơ bản trên đồ thị là phân loại nút, cố gắng phân loại các nút thành 1 vài danh mục được xác định trước. E.g., trong mạng xã hội, cần dự đoán khuynh hướng chính trị của từng người dùng; trong

mạng tương tác protein-protein, cần dự đoán vai trò chức năng của từng protein; trong WWW, cần phân loại các trang web thành các danh mục ngữ nghĩa khác nhau. Để dự đoán hiệu quả, 1 vấn đề quan trọng là phải có các biểu diễn nút thật hiệu quả, yếu tố quyết định phần lớn hiệu suất phân loại nút.

GNNs are neural network architectures specially designed for learning representations of graph-structured data including learning node representations of big graphs (e.g., social networks & WWW) & learning representations of entire graphs (e.g., molecular graphs). In this chap, focus on learning node representations for large-scale graphs & introduce learning whole-graph representations in other chaps. A variety of GNNs have been proposed (Kipf & Welling, 2017b; Veličković et al, 2018; Gilmer et al, 2017; Xhonneux et al, 2020; Liao et al, 2019b; Kipf & Welling, 2016; Veličković et al, 2019). In this chap, comprehensively revisit existing GNNs for node classification including supervised approaches (Sec. 4.2), unsupervised approaches (Sec. 4.3), & a common problem of GNNs for node classification – oversmoothing (Sect. 4.4).

– GNN là kiến trúc mạng nơ-ron được thiết kế đặc biệt để học các biểu diễn dữ liệu có cấu trúc đồ thị, bao gồm học các biểu diễn nút của đồ thị lớn (e.g.: mạng xã hội & WWW) & học các biểu diễn của toàn bộ đồ thị (e.g.: đồ thị phân tử). Trong chương này, chúng tôi tập trung vào việc học các biểu diễn nút cho đồ thị quy mô lớn & giới thiệu về học các biểu diễn toàn bộ đồ thị trong các chương khác. Nhiều loại GNN đã được đề xuất (Kipf & Welling, 2017b; Veličković & cộng sự, 2018; Gilmer & cộng sự, 2017; Xhonneux & cộng sự, 2020; Liao & cộng sự, 2019b; Kipf & Welling, 2016; Veličković & cộng sự, 2019). Trong chương này, chúng tôi sẽ xem xét lại toàn diện các GNN hiện có để phân loại nút bao gồm các phương pháp có giám sát (Phần 4.2), các phương pháp không có giám sát (Phần 4.3) & 1 vấn đề phổ biến của GNN để phân loại nút – làm mịn quá mức (Phần 4.4).

Problem Def. 1st formally define problem of learning node representations for node classification with GNNs. Let $G = (V, E)$ denotes a graph, where V: set of nodes, E: set of edges. $A \in \mathbb{R}^{N\times N}$ represents adjacency matrix, where N: total number of nodes, $X \in \mathbb{R}^{N\times C}$ represents node attribute matrix, where C: number of features for each node. Goal of GNNs: learn effective node representations (denoted as $H \in \mathbb{R}^{N\times F}$, F: dimension of node representations) by combining graph structure information & node attributes, which are further used for node classification.

– Đầu tiên, định nghĩa chính thức bài toán học biểu diễn nút để phân loại nút bằng GNN. Giả sử $G = (V, E)$ là 1 đồ thị, trong đó V: tập các nút, E: tập các cạnh. $A \in \mathbb{R}^{N\times N}$ là ma trận kề, trong đó N: tổng số nút, $X \in \mathbb{R}^{N\times C}$ là ma trận thuộc tính nút, trong đó C: số đặc trưng cho mỗi nút. Mục tiêu của GNN: học các biểu diễn nút hiệu quả (ký hiệu là $H \in \mathbb{R}^{N\times F}$, F: số chiều của biểu diễn nút) bằng cách kết hợp thông tin cấu trúc đồ thị & thuộc tính nút, sau đó được sử dụng để phân loại nút.

- 4.2. Supervised GNNs. In this sect, revisit several representative methods of GNNs for node classification. Focus on supervised methods & introduce unsupervised methods in next sect. Start by introducing a general framework of GNNs & then introduce different variants under this framework.

  – GNN có giám sát. Trong phần này, hãy xem lại 1 số phương pháp GNN tiêu biểu để phân loại nút. Tập trung vào các phương pháp có giám sát & giới thiệu các phương pháp không giám sát trong phần tiếp theo. Bắt đầu bằng cách giới thiệu 1 khuôn khổ chung về GNN & sau đó giới thiệu các biến thể khác nhau trong khuôn khổ này.

  * 4.2.1. General Framework of GNNs. Essential idea of GNNs: iteratively update node representations by combining representations of their neighbors & their own representations. In this sect, introduce a general framework of GNNs in (Xu et al, 2019d). Starting from initial node representaiton $H^0 = X$, in each layer we have 2 important functions:
    1. AGGREGATE, which tries to aggregate information from neighbors of each node
    2. COMBINE, which tries to update node representations by combining aggregated information from neighbors with current node representations.

    – Khung Tổng quát của GNN. Ý tưởng cốt lõi của GNN: cập nhật lặp lại các biểu diễn nút bằng cách kết hợp các biểu diễn của các nút lân cận & các biểu diễn của chính chúng. Trong phần này, giới thiệu 1 khung tổng quát của GNN trong (Xu & cộng sự, 2019d). Bắt đầu từ biểu diễn nút ban đầu $H^0 = X$, trong mỗi lớp, chúng ta có 2 hàm quan trọng:
    1. AGGREGATE, cố gắng tổng hợp thông tin từ các nút lân cận của mỗi nút
    2. COMBINE, cố gắng cập nhật các biểu diễn nút bằng cách kết hợp thông tin tổng hợp từ các nút lân cận với các biểu diễn nút hiện tại.

    Mathematically, can define general framework of GNNs as follows: Initialization $H^0 = X$. For $k \in [K]$

$$a_v^k = AGGREGATE^k\{H_u^{k-1} : u \in N(v)\},\ H_v^k = COMBINE^k\{H_v^{k-1}, a_v^k\},$$

    where $N(v)$: set of neighbors for vth node. Node representations H^K in last layer can be treated as final node representations.

    – Về mặt toán học, có thể định nghĩa khuôn khổ chung của GNN như sau: Khởi tạo $H^0 = X$. Với $k \in [K]$

$$a_v^k = AGGREGATE^k\{H_u^{k-1} : u \in N(v)\},\ H_v^k = COMBINE^k\{H_v^{k-1}, a_v^k\},$$

    trong đó $N(v)$: tập hợp các lân cận cho nút thứ v. Biểu diễn nút H^K ở lớp cuối cùng có thể được coi là biểu diễn nút cuối cùng.

    Once have node representations, they can be used for downstream tasks. Take node classification as an example, label of node v, denoted as $\hat{y}_v$, can be predicted through a Softmax function, i.e.,

$$\hat{y}_v = \text{Softmax}(WH_v^\top),$$

where $W \in \mathbb{R}^{|L| \times F}$, $|L|$: number of labels in output space.
– Sau khi có biểu diễn nút, chúng có thể được sử dụng cho các tác vụ hạ nguồn. Lấy phân loại nút làm e.g., nhãn của nút v, được ký hiệu là $\hat{y}_v$, có thể được dự đoán thông qua hàm Softmax, tức là,

$$\hat{y}_v = \text{Softmax}(W H_v^\top),$$

trong đó $W \in \mathbb{R}^{|L| \times F}$, $|L|$: số nhãn trong không gian đầu ra.
Given a set of labeled nodes, whole model can be trained by minimizing following loss function:

$$O = \frac{1}{n_l} \sum_{i=1}^{n_l} \text{loss}(\hat{y}_i, y_i),$$

where y_i: ground truth label of node i, n_l: number of labeled nodes, $\text{loss}(\cdot, \cdot)$: a loss function e.g. cross-entropy loss function. Whole neural networks can be optimized by minimizing objective function O with backpropagation.
– Với 1 tập hợp các nút được gắn nhãn, toàn bộ mô hình có thể được huấn luyện bằng cách tối thiểu hóa hàm mất mát sau:

$$O = \frac{1}{n_l} \sum_{i=1}^{n_l} \text{loss}(\hat{y}_i, y_i),$$

trong đó y_i: nhãn thực tế của nút i, n_l: số nút được gắn nhãn, $\text{loss}(\cdot, \cdot)$: hàm mất mát, e.g.: hàm mất mát entropy chéo. Toàn bộ mạng nơ-ron có thể được tối ưu hóa bằng cách tối thiểu hóa hàm mục tiêu O với lan truyền ngược.
Above present a general framework of GNNs. Introduce a few most representative instantiations or variants of GNNs in literature.
– Trên đây trình bày 1 khuôn khổ chung về GNN. Giới thiệu 1 số ví dụ hoặc biến thể tiêu biểu nhất của GNN trong tài liệu.

∗ 4.2.2. Graph Convolutional Networks. Start from graph convolutional networks (GCN) (Kipf & Welling, 2017b), which is now most popular GNN architecture due to its simplicity & effectiveness in a variety of tasks & applications. Specifically, node representations in each layer is updated according to following propagation rule: (4.5)

$$H^{k+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^k W^k).$$

$\tilde{A} = A + I$: adjacency matrix of given undirected graph G with self-connections, which allows to incorporate node features itself when updating node representations. $I \in \mathbb{R}^{N \times N}$: identity matrix. $\tilde{D}$: a diagonal matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \sigma(\cdot)$: an activation function e.g. ReLU & Tanh. ReLU active function is widely used, defined as $\text{ReLU}(x) = \max\{0, x\} = x^+$. $W^k \in \mathbb{R}^{F \times F'}$ (F, F': dimensions of node representations in kth, $(k+1)$th layer resp.) is a laywise linear transformation matrix, which will be trained during optimization.
– Mạng Tích chập Đồ thị. Bắt đầu từ mạng tích chập đồ thị (GCN) (Kipf & Welling, 2017b), hiện là kiến trúc GNN phổ biến nhất do tính đơn giản & hiệu quả của nó trong nhiều tác vụ & ứng dụng. Cụ thể, biểu diễn nút trong mỗi lớp được cập nhật theo quy tắc lan truyền sau:

$$H^{k+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^k W^k).$$

$\tilde{A} = A + I$: ma trận kề của đồ thị vô hướng G cho trước với các kết nối tự thân, cho phép kết hợp chính các đặc trưng của nút khi cập nhật biểu diễn nút. $I \in \mathbb{R}^{N \times N}$: ma trận đơn vị. $\tilde{D}$: ma trận đường chéo với $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \sigma(\cdot)$: hàm kích hoạt, e.g.: ReLU & Tanh. Hàm hoạt động của ReLU được sử dụng rộng rãi, được định nghĩa là $\text{ReLU}(x) = \max\{0, x\} = x^+$. $W^k \in \mathbb{R}^{F \times F'}$ (F, F': chiều của biểu diễn nút trong lớp thứ k, tương ứng là $(k+1)$) là ma trận biến đổi tuyến tính từng lớp, sẽ được huấn luyện trong quá trình tối ưu hóa.
Can further dissect (4.5) & understand AGGREGATE & COMBINE function defined in GCN. For a node i, node updating equation can be reformulated as below: (4.6)–(4.7)

$$H_i^k = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} H_j^{k-1} W^k \right),$$

$$H_i^k = \sigma\left(\sum_{j \in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} H_j^{k-1} W^k + \frac{1}{\tilde{D}_i} H_i^{k-1} W^k \right).$$

In (4.7), can see: AGGREGATE function is defined as weighted average of neighbor node representations. Weight of neighbor j is determined by weight of edge between i, j (i.e., A_{ij} normalized by degrees of 2 nodes). COMBINE function is defined as summation of aggregated messages & node representation itself, in which node representation is normalized by its own degree.
– Có thể phân tích sâu hơn (4.5) & hiểu hàm AGGREGATE & COMBINE được định nghĩa trong GCN. Đối với 1 nút i, phương trình cập nhật nút có thể được xây dựng lại như sau: (4.6)–(4.7)

$$H_i^k = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} H_j^{k-1} W^k \right),$$

$$H_i^k = \sigma\left(\sum_{j\in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} H_j^{k-1} W^k + \frac{1}{\tilde{D}_i} H_i^{k-1} W^k\right).$$

Trong (4.7), có thể thấy: Hàm AGGREGATE được định nghĩa là trung bình có trọng số của các biểu diễn nút lân cận. Trọng số của nút lân cận j được xác định bởi trọng số của cạnh giữa i, j (i.e., A_{ij} được chuẩn hóa theo bậc của 2 nút). Hàm COMBINE được định nghĩa là tổng của các thông điệp được tổng hợp & chính biểu diễn nút, trong đó biểu diễn nút được chuẩn hóa theo bậc của chính nó.

Connections with Spectral Graph Convolutions. Discuss connections between GGNs & traditional spectral filters defined on graphs (Defferrard et al, 2016). Spectral convolutions on graphs can be defined as a multiplication of a node-wise signal $\mathbf{x} \in \mathbb{R}^N$ with a convoutional filter $g_\theta = \text{diag}\theta$ ($\theta \in \mathbb{R}^N$: parameter of filter) in Fourier domain. Mathematically, (4.8)

$$g_\theta \star \mathbf{x} = U g_\theta U^\top \mathbf{x}.$$

U represents matrix of eigenvectors of normalized graph Laplacian matrix $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. $L = U\Lambda U^\top$, Λ: a diagonal matrix of eigenvalues, & $U^\top \mathbf{x}$: graph Fourier transform of input signal $\mathbf{x}$. In practice, g_θ can be understood as a function of eigenvalues of normalized graph Laplacian matrix L, i.e., $g_\theta(\Lambda)$. [A lot of rigorous mathematical formulas] p. 45+++

∗ 4.2.3. Graph Attention Networks. In GCNs, for a target node i, importance of a neighbor j is determined by weight of their edge A_{ij} (normalized by their node degrees). However, in practice, input graph may be noisy. Edge weights may not be able to reflect true strength between 2 nodes. As a result, a more principled approach would be to automatically learn importance of each neighbor. Graph Attention Networks (a.k.a. GAT(Veličković et al, 2018)) is built on this idea & try to learn importance of each neighbor based on Attention mechanism (Bahdanau et al, 2015; Vaswani et al, 2017). Attention mechanism has been widely used in a variety of tasks in natural language understanding (e.g., machine translation & question answering) & computer vision (e.g., visual question answering & image captioning). Next, introduce how attention is used in GNNs.

– Graph Attention Networks. Trong GCN, đối với 1 nút đích i, tầm quan trọng của 1 nút lân cận j được xác định bằng trọng số của cạnh A_{ij} của chúng (được chuẩn hóa theo bậc nút của chúng). Tuy nhiên, trên thực tế, đồ thị đầu vào có thể bị nhiễu. Trọng số cạnh có thể không phản ánh được độ mạnh thực sự giữa 2 nút. Do đó, 1 cách tiếp cận có nguyên tắc hơn sẽ là tự động tìm hiểu tầm quan trọng của từng nút lân cận. Graph Attention Networks (hay còn gọi là GAT(Veličković et al, 2018)) được xây dựng dựa trên ý tưởng này & cố gắng tìm hiểu tầm quan trọng của từng nút lân cận dựa trên cơ chế Attention (Bahdanau et al, 2015; Vaswani et al, 2017). Cơ chế Attention đã được sử dụng rộng rãi trong nhiều tác vụ khác nhau trong việc hiểu ngôn ngữ tự nhiên (e.g.: dịch máy & trả lời câu hỏi) & thị giác máy tính (e.g.: trả lời câu hỏi trực quan & chú thích hình ảnh). Tiếp theo, hãy giới thiệu cách sử dụng attention trong GNN.

Graph Attention Layer. Graph attention layer defines how to transfer widen node representations at layer $k-1$ (denoted as $H^{k-1} \in \mathbb{R}^{N\times F}$) to new node representations $H^k \in \mathbb{R}^{N\times F'}$. In order to guarantee sufficient expressive power to transform lower-level node representations to higher-level node representations, a shared linear transformation is applied to every node, denoted as $W \in \mathbb{R}^{F\times F'}$. Afterwards, self-attention is defined on nodes, which measures attention coefficients for any pair of nodes through a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$: (4.14)

$$e_{ij} = a(WH_i^{k-1}, WH_j^{k-1}).$$

e_{ij} indicates relationship strength between node i, j. In this subsect, use H_i^{k-1} to represent a columnwise vector instead of a rowwise vector. For each node, can theoretically allow it to attend to every other node on graph, which however will ignore graph structural information. A more reasonable solution would be only to attend to neighbors for each node. In practice, 1st-order neighbors are only used (including node itself). & to make coefficients comparable across different nodes, attention coefficients are usually normalized with softmax function: (4.15)

$$\alpha_{ij} = \text{Softmax}_j)\{e_{ij}\} = \frac{\exp e_{ij}}{\sum_{l\in N(i)} \exp e_{il}}.$$

For a node i, α_{ij} essentially defines a multinomial distribution over neighbors, which can also be interpreted as transition probability from node i to each of its neighbors.

– **Lớp Chú ý Đồ thị.** Lớp chú ý đồ thị xác định cách chuyển các biểu diễn nút mở rộng ở lớp $k-1$ (ký hiệu là $H^{k-1} \in \mathbb{R}^{N\times F}$) sang các biểu diễn nút mới $H^k \in \mathbb{R}^{N\times F'}$. Để đảm bảo đủ sức mạnh biểu đạt để chuyển đổi các biểu diễn nút cấp thấp hơn sang các biểu diễn nút cấp cao hơn, 1 phép biến đổi tuyến tính chung được áp dụng cho mọi nút, ký hiệu là $W \in \mathbb{R}^{F\times F'}$. Sau đó, sự tự chú ý được định nghĩa trên các nút, đo lường hệ số chú ý cho bất kỳ cặp nút nào thông qua cơ chế chú ý chung $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$: (4.14)

$$e_{ij} = a(WH_i^{k-1}, WH_j^{k-1}).$$

e_{ij} biểu thị cường độ mối quan hệ giữa nút i, j. Trong tiểu mục này, sử dụng H_i^{k-1} để biểu diễn 1 vectơ theo cột thay vì vectơ theo hàng. Đối với mỗi nút, về mặt lý thuyết, có thể cho phép nó chú ý đến mọi nút khác trên đồ thị, tuy nhiên điều này sẽ bỏ qua thông tin cấu trúc đồ thị. 1 giải pháp hợp lý hơn sẽ chỉ chú ý đến các nút lân cận cho mỗi nút. Trong thực tế, các nút lân cận bậc 1 chỉ được sử dụng (bao gồm cả chính nút đó). & để làm cho các hệ số có thể so sánh được giữa các nút khác nhau, các hệ số chú ý thường được chuẩn hóa bằng hàm softmax:

$$\alpha_{ij} = \text{Softmax}_j(\{e_{ij}\}) = \frac{\exp e_{ij}}{\sum_{l\in N(i)} \exp e_{il}}.$$

Đối với 1 nút i, α_{ij} về cơ bản xác định 1 phân phối đa thức trên các nút lân cận, cũng có thể được hiểu là xác suất chuyển tiếp từ nút i đến từng nút lân cận của nó.
In the work by Veličković et al (2018), attention mechanism a is defined as a single-layer feedforward neural network including a linear transformation with weight vector $W_2 \in \mathbb{R}^{1\times 2F'}$ & a LeakyReLU nonlinear activation function (with negative input slope $\alpha = 0.2$). More specifically, can calculate attention coefficients with following architecture:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(W_2[WH_i^{k-1}||WH_j^{k-1}]))}{\sum_{l\in N(i)} \exp(\text{LeakyReLU}(W_2[WH_i^{k-1}||WH_l^{k-1}]))}$$

where || represents operation of concatenating 2 vectors. New node representation is a linear combination of neighboring node representations with weights determined by attention coefficients (with a potential nonlinear transformation), i.e., (4.17)

$$H_i^k = \sigma\left(\sum_{j\in N(i)} \alpha_{ij} W H_j^{k-1}\right).$$

– Trong công trình của Veličković et al (2018), cơ chế chú ý a được định nghĩa là mạng nơ-ron truyền thẳng 1 lớp bao gồm phép biến đổi tuyến tính với vectơ trọng số $W_2 \in \mathbb{R}^{1\times 2F'}$ & hàm kích hoạt phi tuyến tính LeakyReLU (với độ dốc đầu vào âm $\alpha = 0,2$). Cụ thể hơn, có thể tính toán hệ số chú ý với kiến trúc sau:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(W_2[WH_i^{k-1}||WH_j^{k-1}]))}{\sum_{l\in N(i)} \exp(\text{LeakyReLU}(W_2[WH_i^{k-1}||WH_l^{k-1}]))}$$

trong đó || biểu diễn phép toán nối 2 vectơ. Biểu diễn nút mới là tổ hợp tuyến tính của các biểu diễn nút lân cận với trọng số được xác định bởi hệ số chú ý (có khả năng biến đổi phi tuyến tính), tức là (4.17)

$$H_i^k = \sigma\left(\sum_{j\in N(i)} \alpha_{ij} W H_j^{k-1}\right).$$

Multihead Attention. In practice, instead of only using 1 single attention mechanism, *multihead attention* can be used, each of which determines a different similarity function over nodes. For each attention head, can independently obtian a new node representation according to (4.17). Final node representation will be a concatenation of node representations learned by different attention heads. Mathematically, we have (4.18)

$$H_i^k = ||_{t=1}^T \sigma\left(\sum_{j\in N(i)} \alpha_{ij}^t W^t H_j^{k-1}\right),$$

where T: total number of attention heads, α_{ij}^t is attention coefficient calculated from tth attention head, W^t is linear transformation matrix of tth attention head.
– **Multihead Attention.** Trên thực tế, thay vì chỉ sử dụng 1 cơ chế chú ý duy nhất, có thể sử dụng *multihead attention*, mỗi cơ chế xác định 1 hàm tương tự khác nhau trên các nút. Với mỗi đầu chú ý, có thể độc lập tạo ra 1 biểu diễn nút mới theo (4.17). Biểu diễn nút cuối cùng sẽ là 1 chuỗi các biểu diễn nút được học bởi các đầu chú ý khác nhau. Về mặt toán học, ta có (4.18)

$$H_i^k = ||_{t=1}^T \sigma\left(\sum_{j\in N(i)} \alpha_{ij}^t W^t H_j^{k-1}\right),$$

trong đó T: tổng số đầu chú ý, α_{ij}^t là hệ số chú ý được tính từ đầu chú ý thứ t, W^t là ma trận biến đổi tuyến tính của đầu chú ý thứ t.
1 thing mentioned in paper by Veličković et al (2018): in final layer, when trying to combine node representations from different attention heads, instead of using operation concatenation, other pooling techniques could be used, e.g., simply taking average node representations from different attention heads. (4.19)

$$H_i^k = \sigma\left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in N(i)} \alpha_{ij}^t W^t H_j^{k-1}\right).$$

– 1 điều được đề cập trong bài báo của Veličković & cộng sự (2018): ở lớp cuối cùng, khi cố gắng kết hợp các biểu diễn nút từ các đầu chú ý khác nhau, thay vì sử dụng phép nối toán tử, có thể sử dụng các kỹ thuật gộp khác, ví dụ, chỉ cần lấy các biểu diễn nút trung bình từ các đầu chú ý khác nhau. (4.19)

$$H_i^k = \sigma\left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in N(i)} \alpha_{ij}^t W^t H_j^{k-1}\right).$$

∗ 4.2.4. Neural Message Passing Networks. Another very popular GNN architecture: Neural Message Passing Network (MPNN) (Gilmer et al, 2017), which is originally proposed for learning molecular graph representations. However, MPNN is actually very general, provides a general framework of GNNs, & could be used for task of node classification as well. Essential idea of MPNN is formulating existing GNNs as a general framework of neural message passing among nodes. In MPNNs, there are 2 important functions including *Message* & *Updating* function: (4.20)–(4.21)

$$m_i^k = \sum_{i \in N(j)} M_k(H_i^{k-1}, H_j^{k-1}, e_{ij}),$$
$$H_i^k = U_k(H_i^{k-1}, m_i^k).$$

$M_k(\cdot,\cdot,\cdot)$ defines message between node i, j in kth layer, which depends on 2 node representations & information of their edge. U_k is node updating function in kth layer which combines aggregated messages from neighbors & node representation itself. Can see: MPNN framework is very similar to general framework introduced in Sect. 4.2.1. AGGREGATE function defined here is simply a summation of all the messages from neighbors. COMBINE function is same as node Updating function.

– Mạng Truyền Thông Tin Nơ-ron. 1 kiến trúc GNN rất phổ biến khác: Mạng Truyền Thông Tin Nơ-ron (MPNN) (Gilmer & cộng sự, 2017), ban đầu được đề xuất để học các biểu diễn đồ thị phân tử. Tuy nhiên, MPNN thực sự rất tổng quát, cung cấp 1 khuôn khổ chung cho GNN, & cũng có thể được sử dụng cho nhiệm vụ phân loại nút. Ý tưởng cốt lõi của MPNN là xây dựng các GNN hiện có như 1 khuôn khổ chung cho việc truyền thông tin nơ-ron giữa các nút. Trong MPNN, có 2 hàm quan trọng bao gồm *Message* & *Updating*: (4.20)–(4.21)

$$m_i^k = \sum_{i \in N(j)} M_k(H_i^{k-1}, H_j^{k-1}, e_{ij}),$$
$$H_i^k = U_k(H_i^{k-1}, m_i^k).$$

$M_k(\cdot,\cdot,\cdot)$ định nghĩa thông điệp giữa nút i, j trong lớp k, phụ thuộc vào biểu diễn của 2 nút & thông tin về cạnh của chúng. U_k là hàm cập nhật nút trong lớp k, kết hợp các thông điệp được tổng hợp từ các nút lân cận & chính biểu diễn nút đó. Có thể thấy: Khung MPNN rất giống với khung chung được giới thiệu trong Mục 4.2.1. Hàm AGGREGATE được định nghĩa ở đây đơn giản là tổng hợp tất cả các thông điệp từ các nút lân cận. Hàm COMBINE giống với hàm Cập nhật nút.

∗ 4.2.5. Continuous GNNs. Above GNNs iteratively update node representations with different kinds of graph convolutional layers. Essentially, these approaches model discrete dynamics of node representations with GNNs. Xhonneux et al (2020) proposed continuous graph neural networks (CGNNs), which generalizes existing GNNs with discrete dynamics to continuous settings, i.e., trying to model continuous dynamics of node representations. Key idea is how to characterize continuous dynamics of node representations, i.e., derivatives of node representation w.r.t. time. CGNN model is inspired by diffusion-based models on graphs e.g. PageRank & epidemic models on social networks. Derivatives of node representations are defined as a combination of node representation itself, representations of its neighbors, & initial status of nodes. Specifically, 2 different variants of node dynamics are introduced. 1st model assumes that different dimensions of node representations (a.k.a. feature channels) are independent; 2nd model is more flexible, which allows different feature channels to interact with each other. Next, give a detailed introduction to each of 2 models.

– GNN liên tục. Các GNN ở trên cập nhật lặp lại các biểu diễn nút với các loại lớp tích chập đồ thị khác nhau. Về cơ bản, các phương pháp này mô hình hóa động lực rời rạc của các biểu diễn nút bằng GNN. Xhonneux & cộng sự (2020) đã đề xuất mạng nơ-ron đồ thị liên tục (CGNN), khái quát hóa các GNN hiện có với động lực rời rạc thành các thiết lập liên tục, tức là cố gắng mô hình hóa động lực liên tục của các biểu diễn nút. Ý tưởng chính là cách mô tả động lực liên tục của các biểu diễn nút, tức là các đạo hàm của biểu diễn nút theo thời gian. Mô hình CGNN lấy cảm hứng từ các mô hình dựa trên sự khuếch tán trên đồ thị, ví dụ: PageRank & các mô hình dịch bệnh trên mạng xã hội. Các đạo hàm của các biểu diễn nút được định nghĩa là sự kết hợp của chính biểu diễn nút, các biểu diễn của các nút lân cận của nó, & trạng thái ban đầu của các nút. Cụ thể, 2 biến thể khác nhau của động lực nút được giới thiệu. Mô hình thứ nhất giả định rằng các chiều khác nhau của biểu diễn nút (hay còn gọi là kênh đặc trưng) là độc lập; Mô hình thứ 2 linh hoạt hơn, cho phép các kênh tính năng khác nhau tương tác với nhau. Tiếp theo, hãy giới thiệu chi tiết về từng mô hình trong 2 mô hình.

Remark 3. *In this part, instead of using original adjacency matrix A, use following regularized matrix for characterizing graph structure (4.22)*

$$A := \frac{\alpha}{2}(I + D^{-1/2} A D^{-1/2}),$$

where $\alpha \in (0, 1)$ is a hyperparameter. D: degree matrix of original adjacency matrix A. With new regularized adjacency matrix A, eigenvalues of A will lie in $[0, \alpha]$, which will make A^k converges to 0 when we increase power of k.

– Trong phần này, thay vì sử dụng ma trận kề ban đầu A, hãy sử dụng ma trận chính quy sau để mô tả cấu trúc đồ thị (4.22)

$$A := \frac{\alpha}{2}(I + D^{-1/2} A D^{-1/2}),$$

trong đó $\alpha \in (0, 1)$ là 1 siêu tham số. D: ma trận bậc của ma trận kề ban đầu A. Với ma trận kề chính quy mới A, các giá trị riêng của A sẽ nằm trong $[0, \alpha]$, điều này sẽ làm cho A^k hội tụ về 0 khi ta tăng lũy thừa của k.

Model 1: Independent Feature Channels. As different nodes in a graph are interconnected, a natural solution to model dynamic of each feature channel should be taking graph structure into consideration, which allows information to propagate across different nodes. Motivated by existing diffusion-based methods on graphs e.g. PageRank (Page et al, 1999) & label propagation (Zhou et al, 2004), which defines discrete propagation of node representations (or signals on nodes) with following step-wise propagation equations: (4.23)

$$H^{k+1} = AH^k + H^0,$$

where $H^0 = X$ or output of an encoder on input feature X. Intuitively, at each step, new node representation is a linear combination of its neighboring node representations as well as initial node features. Such a mechanism allows to model information propagation on graph without forgetting initial node features. Can unroll (4.23) & explicitly derive node representations at kth step: (4.24)

$$H^k = \left(\sum_{i=0}^{k} A^i\right) H^0 = (A - I)^{-1}(A^{k+1} - I)H^0.$$

– **Mô hình 1: Kênh Tính năng Độc lập.** Vì các nút khác nhau trong đồ thị được kết nối với nhau, 1 giải pháp tự nhiên để mô hình hóa động lực của mỗi kênh tính năng nên được xem xét đến cấu trúc đồ thị, cho phép thông tin lan truyền qua các nút khác nhau. Được thúc đẩy bởi các phương pháp dựa trên khuếch tán hiện có trên đồ thị, ví dụ như PageRank (Page & cộng sự, 1999) & lan truyền nhãn (Zhou & cộng sự, 2004), định nghĩa sự lan truyền rời rạc của các biểu diễn nút (hoặc tín hiệu trên các nút) với các phương trình lan truyền từng bước sau: (4.23)

$$H^{k+1} = AH^k + H^0,$$

trong đó $H^0 = X$ hoặc đầu ra của bộ mã hóa trên tính năng đầu vào X. 1 cách trực quan, tại mỗi bước, biểu diễn nút mới là 1 tổ hợp tuyến tính của các biểu diễn nút lân cận cũng như các tính năng nút ban đầu. Cơ chế như vậy cho phép mô hình hóa sự lan truyền thông tin trên đồ thị mà không bỏ sót các tính năng nút ban đầu. Có thể mở rộng (4.23) & suy ra biểu diễn nút 1 cách rõ ràng tại bước thứ k: (4.24)

$$H^k = \left(\sum_{i=0}^{k} A^i\right) H^0 = (A - I)^{-1}(A^{k+1} - I)H^0.$$

As above equation effectively models discrete dynamics of node representations, CGNN model further extended it to continuous setting, which replaces discrete time step k to a continuous variable $t \in \mathbb{R}_0^+$. Specifically, it has been shown: (4.24) is a discretization of following ODE: (4.25)

$$\frac{dH^t}{dt} = \log AH^t + X,$$

with initial value $H^0 = (\log A)^{-1}(A - I)X$, where X is initial node features or output of an encoder applied to it. Do not provide the proof here. More details can be referred to the original paper (Xhonneux et al, 2020). In (4.25), as $\log A$ is intractable to compute in practice, it is approximated with 1st-order of Taylor expansion, i.e., $\log A \approx A - I$. By integrating all these information, have following ODE equation: (4.26)

$$\frac{dH^t}{dt} = (A - I)H^t + X,$$

with initial value $H^0 = X$, which is 1st variant of CGNN model.
– Vì phương trình trên mô hình hóa hiệu quả động lực học rời rạc của biểu diễn nút, mô hình CGNN tiếp tục mở rộng nó thành thiết lập liên tục, thay thế bước thời gian rời rạc k thành biến liên tục $t \in \mathbb{R}_0^+$. Cụ thể, nó đã được chứng minh: (4.24) là 1 phép rời rạc của ODE sau: (4.25)

$$\frac{dH^t}{dt} = \log AH^t + X,$$

với giá trị ban đầu $H^0 = (\log A)^{-1}(A - I)X$, trong đó X là các đặc trưng nút ban đầu hoặc đầu ra của bộ mã hóa được áp dụng cho nó. Không cung cấp bằng chứng ở đây. Chi tiết hơn có thể được tham khảo trong bài báo gốc (Xhonneux & cộng sự, 2020). Trong (4.25), do $\log A$ khó tính toán trong thực tế, nó được xấp xỉ bằng khai triển Taylor bậc 1, tức là $\log A \approx A - I$. Bằng cách tích phân tất cả các thông tin này, ta có phương trình ODE sau: (4.26)

$$\frac{dH^t}{dt} = (A - I)H^t + X,$$

với giá trị ban đầu $H^0 = X$, là biến thể thứ nhất của mô hình CGNN.
CGNN model is actually very intuitive, which has a nice connection with traditional epidemic model, which aims at studying dynamics of infection in a population. For epidemic model, it usually assumes: infection of people will be affected by 3 different factors including infection from neighbors, natural recovery, & natural characteristics of people. If

treat H^t as number of people infected at time t, then these 3 factors can be naturally modeled by 3 terms in (4.26): AH^t for infection from neighbors, $-H^t$ for natural recovery, & last one X for natural characteristics of people.
– Mô hình CGNN thực sự rất trực quan, có mối liên hệ chặt chẽ với mô hình dịch tễ truyền thống, vốn nhắm đến việc nghiên cứu động lực lây nhiễm trong quần thể. Đối với mô hình dịch tễ, nó thường giả định: sự lây nhiễm của con người sẽ bị ảnh hưởng bởi 3 yếu tố khác nhau, bao gồm lây nhiễm từ hàng xóm, phục hồi tự nhiên, & đặc điểm tự nhiên của con người. Nếu coi H^t là số người bị nhiễm tại thời điểm t, thì 3 yếu tố này có thể được mô hình hóa tự nhiên bằng 3 số hạng trong (4.26): AH^t cho lây nhiễm từ hàng xóm, $-H^t$ cho phục hồi tự nhiên, & X cho đặc điểm tự nhiên của con người.
Model 2: Modeling Interaction of Feature Channels. Above model assumes different node feature channels are independent with each other, which is a very strong assumption & limits capacity of model. Inspired by success of a linear variant of GNNs (i.e., Simple GCN (Wu et al, 2019a)), a more powerful discrete node dynamic model is proposed, which allows different feature channels to interact with each other as (4.27)

$$H^{k+1} = AH^kW + H^0,$$

where $W \in \mathbb{R}^{F\times F}$ is a weight matrix used to model interactions between different feature channels. Similarly, can also extend above discrete dynamics into continuous case, yielding following equation: (4.28)

$$\frac{dH^t}{dt} = (A - I)H^t + H^t(W - I) + X,$$

with initial value being $H^0 = X$. This is 2nd variant of CGNN with trainable weights. Similar form of ODEs defined in (4.28) has been studied in literature of control theory, known as Sylvester differential equation (Locatelli & Sieniutycz, 2002). 2 matrices $A - I, W - I$ characterize natural solution of system while X is information provided to system to drive system into desired state.
– **Mô hình 2: Mô hình hóa Tương tác của các Kênh Đặc trưng.** Mô hình trên giả định các kênh đặc trưng nút khác nhau độc lập với nhau, đây là 1 giả định rất mạnh & hạn chế khả năng của mô hình. Lấy cảm hứng từ thành công của 1 biến thể tuyến tính của GNN (tức là GCN Đơn giản (Wu & cộng sự, 2019a)), 1 mô hình động nút rời rạc mạnh mẽ hơn được đề xuất, cho phép các kênh đặc trưng khác nhau tương tác với nhau như (4.27)

$$H^{k+1} = AH^kW + H^0,$$

trong đó $W \in \mathbb{R}^{F\times F}$ là ma trận trọng số được sử dụng để mô hình hóa tương tác giữa các kênh đặc trưng khác nhau. Tương tự, cũng có thể mở rộng động lực học rời rạc trên sang trường hợp liên tục, thu được phương trình sau: (4.28)

$$\frac{dH^t}{dt} = (A - I)H^t + H^t(W - I) + X,$$

với giá trị ban đầu là $H^0 = X$. Đây là biến thể thứ 2 của CGNN với các trọng số có thể huấn luyện được. Dạng tương tự của ODE được định nghĩa trong (4.28) đã được nghiên cứu trong các tài liệu về lý thuyết điều khiển, được gọi là phương trình vi phân Sylvester (Locatelli & Sieniutycz, 2002). 2 ma trận $A - I, W - I$ đặc trưng cho nghiệm tự nhiên của hệ thống trong khi X là thông tin được cung cấp cho hệ thống để đưa hệ thống vào trạng thái mong muốn.
Discussion. Proposed CGNN has multiple nice properties:

1. Recent work has shown that if we increase number of layers K in discrete graph neural networks, learned node representations tend to have problem of over-smoothing & hence lose power of expressiveness. On contrary, continuous graph neural networks are able to train very deep graph neural networks & are experimentally robust to arbitrarily chosen integration time.
2. For some of tasks on graphs, critical to model long-range dependency between nodes, which requires training deep GNNs. Existing discrete GNNs fail to train very deep GNNs due to oversmoothing problem. CGNNs are able to efficiently model long-range dependency between nodes thanks to stability w.r.t. time.
3. Hyperparameter α is very important, which controls rate of diffusion. Specifically, it controls rate at which high-order powers of regularized matrix A vanishes. In work proposed by (Xhonneux et al, 2020), the authors proposed to learn a different value of α for each node, which hence allows to choose best diffusion rates for different nodes.

– **Thảo luận.** CGNN được đề xuất có nhiều đặc tính tốt:

1. Các nghiên cứu gần đây đã chỉ ra rằng nếu chúng ta tăng số lớp K trong mạng nơ-ron đồ thị rời rạc, các biểu diễn nút đã học có xu hướng gặp vấn đề làm mịn quá mức & do đó mất đi khả năng biểu đạt. Ngược lại, mạng nơ-ron đồ thị liên tục có thể huấn luyện mạng nơ-ron đồ thị rất sâu & có khả năng thực nghiệm mạnh mẽ với thời gian tích hợp được chọn tùy ý.
2. Đối với 1 số tác vụ trên đồ thị, việc mô hình hóa sự phụ thuộc tầm xa giữa các nút là rất quan trọng, đòi hỏi phải huấn luyện các GNN sâu. Các GNN rời rạc hiện có không thể huấn luyện các GNN rất sâu do vấn đề làm mịn quá mức. CGNN có thể mô hình hóa hiệu quả sự phụ thuộc tầm xa giữa các nút nhờ tính ổn định theo thời gian.
3. Siêu tham số α rất quan trọng, nó kiểm soát tốc độ khuếch tán. Cụ thể, nó kiểm soát tốc độ mà lũy thừa bậc cao của ma trận chính quy A triệt tiêu. Trong công trình được đề xuất bởi (Xhonneux & cộng sự, 2020), các tác giả đề xuất tìm hiểu 1 giá trị α khác nhau cho mỗi nút, từ đó cho phép chọn tốc độ khuếch tán tốt nhất cho các nút khác nhau.

∗ 4.2.6. Multi-Scale Spectral Graph Convolutional Networks. Recall 1-layer graph convolution operator used in GCNs (Kipf & Welling, 2017b) $H = LHW$, where $L = D^{-1/2}\tilde{A}D^{-1/2}$. Here drop superscript of layer index to avoid clash with notation of matrix power. There are 2 main issues with this simple graph convolution formulation. 1st, 1 such graph convolutional layer would only propagate information from any node to its nearest neighbors, i.e., neighboring nodes that are 1-hop away. If one would like to propagate information to M-hop away neighbors, one has to either stack M graph convolutional layers or compute graph convolution with Mth power of graph Laplacian, i.e., $H = \sigma(L^M HW)$. When M is large, solution of stacking layers would make whole GCN model very deep, thus causing problems in learning like vanishing gradient. This is similar to what people experienced in training very deep feedforward neural networks. For matrix power solution, naively computing Mth power of graph Laplacian is also very costly (e.g., time complexity is $O(N^{3(M-1)})$ for graphs with N nodes). 2nd, there are no learnable parameters in GCNs associated with graph Laplacian L (corresponding to connectivities/structures). Only learnable parameter W is a linear transform applied to every node simultaneously which is not aware of structures. Note: typically associate learnable weights on edges while applying convolution applied to regular graphs like grids (e.g., applying 2D convolution to images). [NQBH: see flood fill algorithm for 2D grid-like graphs] This would greatly improve expressiveness of model. However, not clear: how one can add learnable parameters to graph Laplacian L since its size varies from graph to graph.

– Mạng tích chập đồ thị phổ đa thang. Nhắc lại toán tử tích chập đồ thị 1 lớp được sử dụng trong GCN (Kipf & Welling, 2017b) $H = LHW$, trong đó $L = D^{-1/2}\tilde{A}D^{-1/2}$. Ở đây bỏ chỉ số mũ của chỉ số lớp để tránh xung đột với ký hiệu lũy thừa ma trận. Có 2 vấn đề chính với công thức tích chập đồ thị đơn giản này. Thứ nhất, 1 lớp tích chập đồ thị như vậy sẽ chỉ truyền thông tin từ bất kỳ nút nào đến các nút lân cận gần nhất của nó, tức là các nút lân cận cách xa 1 bước nhảy. Nếu muốn truyền thông tin đến các nút lân cận cách xa M bước nhảy, người ta phải xếp chồng M lớp tích chập đồ thị hoặc tính tích chập đồ thị với lũy thừa M của đồ thị Laplacian, tức là $H = \sigma(L^M HW)$. Khi M lớn, giải pháp của các lớp xếp chồng sẽ làm cho toàn bộ mô hình GCN rất sâu, do đó gây ra các vấn đề trong quá trình học như biến mất gradient. Điều này tương tự như những gì mọi người đã trải qua khi huấn luyện các mạng nơ-ron truyền thẳng rất sâu. Đối với giải pháp ma trận, việc tính toán lũy thừa M của đồ thị Laplacian 1 cách ngây thơ cũng rất tốn kém (ví dụ: độ phức tạp thời gian là $O(N^{3(M-1)})$ đối với đồ thị có N nút). Thứ hai, không có tham số có thể học được nào trong GCN liên kết với đồ thị Laplacian L (tương ứng với các cấu trúc kết nối/). Chỉ có tham số có thể học được W là 1 phép biến đổi tuyến tính được áp dụng đồng thời cho mọi nút mà không nhận biết được các cấu trúc. Lưu ý: thường liên kết các trọng số có thể học được trên các cạnh khi áp dụng tích chập được áp dụng cho các đồ thị thông thường như lưới (ví dụ: áp dụng tích chập 2D cho hình ảnh). [NQBH: xem thuật toán điền đầy cho đồ thị dạng lưới 2D] Điều này sẽ cải thiện đáng kể khả năng biểu đạt của mô hình. Tuy nhiên, vẫn chưa rõ: làm thế nào để thêm các tham số có thể học được vào đồ thị Laplacian L vì kích thước của nó thay đổi tùy theo từng đồ thị.

To over these 2 problems, authors propose Lanczos Networks in (Liao et al, 2019b). Given the graph Laplacian matrix L [here assume a symmetric graph Laplacian matrix. If it is nonsymmetric (e.g., for directed graphs), one can resort to Arnoldi algorithm.] & node features X, one 1st uses M-step Lanczos algorithm (Lanczos, 1950) (listed in Algorithm 1: Lanczos algorithm.) to compute an orthogonal matrix Q & a symmetric tridiagonal matrix T, s.t. $Q^\top LQ = T$. Denote $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_M]$ where coumn vector $\mathbf{q}_i$ is ith Lanczos vector. Note M could be much smaller than number of nodes N. T is illustrated as below (4.29) [tridiagonal matrix $[\beta_1, \ldots, \beta_{M-1}], [\gamma_1, \ldots, \gamma_M], [\beta_1, \ldots, \beta_{M-1}]$. After obtaining tridiagonal matrix T, can compute Ritz values & Ritz vectors which approximate top eigenvalues & eigenvectors of L by diagonalizing matrix T as $T = BRB^\top$, where $K \times K$ diagonal matrix R contains Ritz values & $B \in \mathbb{R}^{K \times K}$ is an orthogonal matrix. Here top means ranking eigenvalues by their magnitudes in a descending order. This can be implemented via general eigendecomposition or some fast decomposition methods specialized for tridiagonal matrices. Now have a low rank approximation of graph Laplacian matrix $L \approx VRV^\top$, where $V = QB$. Denoting column vectors of V as $\{\mathbf{v}_1, \ldots, \mathbf{v}_M\}$, can compute multiscale graph convolution as (4.30)

$$H = \hat{L}HW,$$
$$\hat{L} = \sum_{m=1}^{M} f_\theta(r_m^{I_1}, r_m^{I_2}, \ldots, r_m^{I_u}) v_m v_m^\top,$$

where $\{I_1, \ldots, I_u\}$ is set of scale/range parameters which determine how many hops (or how far) one would like to propagate information over graph. E.g., one could easily set $\{I_1 = 50, I_2 = 100\}$ ($u = 2$ in this case) to consider situations of propagating 50 & 100 steps resp. Note one only needs to compute scalar power rather than original matrix power. Overall complexity of Lanczos algorithm in our context is $O(MN^2)$ which makes whole algorithm much more efficient than naively computing matrix power. Moreover, f_θ is a learnable spectral filter paramterized by θ & can be applied to graphs with varying sizes since we decouple graph size & input size of f_θ. f_θ directly acts on graph Laplacian & greatly improves expressiveness of model.

– Để giải quyết 2 vấn đề này, các tác giả đề xuất Mạng Lanczos trong (Liao et al, 2019b). Cho đồ thị ma trận Laplacian L [ở đây giả sử 1 đồ thị đối xứng ma trận Laplacian. Nếu nó không đối xứng (ví dụ, đối với đồ thị có hướng), người ta có thể dùng đến thuật toán Arnoldi.] & các đặc trưng nút X, trước tiên người ta sử dụng thuật toán Lanczos M-bước (Lanczos, 1950) (được liệt kê trong Thuật toán 1: Thuật toán Lanczos.) để tính toán ma trận trực giao Q & ma trận đối xứng ba đường chéo T, s.t. $Q^\top LQ = T$. Ký hiệu $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_M]$ trong đó vectơ số $\mathbf{q}_i$ là vectơ Lanczos thứ i. Lưu ý M có thể nhỏ hơn nhiều so với số nút N. T được minh họa như sau (4.29) [ma trận ba đường chéo $[\beta_1, \ldots, \beta_{M-1}], [\gamma_1, \ldots, \gamma_M], [\beta_1, \ldots, \beta_{M-1}]$. Sau khi thu được ma trận ba đường chéo T, có thể tính các giá trị Ritz & vectơ Ritz xấp xỉ các giá trị riêng trên & vectơ riêng của L bằng cách chéo hóa ma trận T thành $T = BRB^\top$, trong đó $K \times K$ ma trận đường chéo R chứa các giá trị

Ritz & $B \in \mathbb{R}^{K\times K}$ là 1 ma trận trực giao. Ở đây, top có nghĩa là xếp hạng các giá trị riêng theo độ lớn của chúng theo thứ tự giảm dần. Điều này có thể được thực hiện thông qua phân tích riêng tổng quát hoặc 1 số phương pháp phân tích nhanh chuyên biệt cho ma trận ba đường chéo. Bây giờ, chúng ta có 1 phép xấp xỉ bậc thấp của ma trận Laplacian đồ thị $L \approx VRV^\top$, trong đó $V = QB$. Ký hiệu các vectơ cột của V là $\{\mathbf{v}_1, \ldots, \mathbf{v}_M\}$, có thể tính toán tích chập đồ thị đa thang như (4.30)

$$H = \hat{L}HW,$$
$$\hat{L} = \sum_{m=1}^{M} f_\theta(r_m^{I_1}, r_m^{I_2}, \ldots, r_m^{I_u}) v_m v_m^\top,$$

trong đó $\{I_1, \ldots, I_u\}$ là tập hợp các tham số phạm vi thang/quy mô xác định số bước nhảy (hoặc khoảng cách) mà người ta muốn truyền thông tin qua đồ thị. Ví dụ, ta có thể dễ dàng đặt $\{I_1 = 50, I_2 = 100\}$ ($u = 2$ trong trường hợp này) để xét các trường hợp lan truyền 50 & 100 bước tương ứng. Lưu ý, ta chỉ cần tính lũy thừa vô hướng thay vì lũy thừa ma trận gốc. Độ phức tạp tổng thể của thuật toán Lanczos trong bối cảnh của chúng ta là $O(MN^2)$, điều này làm cho toàn bộ thuật toán hiệu quả hơn nhiều so với việc tính toán lũy thừa ma trận 1 cách ngây thơ. Hơn nữa, f_θ là 1 bộ lọc phổ có thể học được được tham số hóa bởi θ & có thể được áp dụng cho các đồ thị có kích thước khác nhau vì chúng ta tách rời kích thước đồ thị & kích thước đầu vào của f_θ. f_θ tác động trực tiếp lên đồ thị Laplacian & cải thiện đáng kể tính biểu đạt của mô hình.

Although Lanczos algorithm provides an efficient way to approximately compute arbitrary powers of graph Laplacian, it is still a low-rank approximation which may lose certain information (e.g., high frequency one). To alleviate problem, one can further do vanilla graph convolution with small scale parameters like $H = L^S HW$ where S could be small integers like 2 or 3. Resultant representation can be concatenated with one obtained from longer scale/range graph convolution in (4.30). Relying on above design, one could add nonlinearities & stack multiple such layers to build a deep graph convolutional network (namely Lanczos Networks) just like GCNs. Overall inference procedure of Lanczos Networks is shown in Fig. 4.1: Inference procedure of Lanczos Networks. Approximated top eigenvalues $\{r_k\}$ & eigenvectors $\{\mathbf{v}_k\}$ are computed by Lanczos algorithm. Note: this step is only needed once per graph. Long range/scale (top blocks) graph convolutions are efficiently computed by low-rank approximation of graph Laplacian. One can control ranges (i.e., exponent of eigenvalues) as hyperparameters. Learnable spectral filters are applied to approximated top eigenvalues $\{r_k\}$. Short range/scale (bottom blocks) graph convolution is same as GCNs. This method demonstrates strong empirical performances on a wide variety of tasks/benchmarks including molecular property prediction in quantum chemistry & document classification in citation networks. it just requires slight modifications to implementation of original GCNs. Nevertheless, if input graph is extremely large (e.g., some large social network), Lanczos algorithm itself would be a computational bottleneck. How to improve this model in such a problem context would be an open question.

– Mặc dù thuật toán Lanczos cung cấp 1 cách hiệu quả để tính toán xấp xỉ các lũy thừa Laplacian tùy ý của đồ thị, nhưng nó vẫn là 1 phép xấp xỉ hạng thấp có thể mất 1 số thông tin nhất định (ví dụ: thông tin tần số cao). Để giảm bớt vấn đề này, người ta có thể thực hiện thêm tích chập đồ thị vani với các tham số quy mô nhỏ như $H = L^S HW$ trong đó S có thể là các số nguyên nhỏ như 2 hoặc 3. Biểu diễn kết quả có thể được nối với biểu diễn thu được từ tích chập đồ thị phạm vi /quy mô dài hơn trong (4.30). Dựa vào thiết kế trên, người ta có thể thêm các phi tuyến tính & xếp chồng nhiều lớp như vậy để xây dựng 1 mạng tích chập đồ thị sâu (cụ thể là Mạng Lanczos) giống như GCN. Quy trình suy luận tổng thể của Mạng Lanczos được thể hiện trong Hình 4.1: Quy trình suy luận của Mạng Lanczos. Các giá trị riêng đỉnh xấp xỉ $\{r_k\}$ & các vectơ riêng $\{\mathbf{v}_k\}$ được tính bằng thuật toán Lanczos. Lưu ý: bước này chỉ cần thực hiện 1 lần cho mỗi đồ thị. Tích chập đồ thị phạm vi dài / (các khối trên cùng) được tính toán hiệu quả bằng phép xấp xỉ hạng thấp của Laplacian đồ thị. Người ta có thể kiểm soát các phạm vi (tức là số mũ của các trị riêng) dưới dạng siêu tham số. Các bộ lọc phổ có thể học được được áp dụng cho các trị riêng đỉnh $\{r_k\}$ xấp xỉ. Tích chập đồ thị phạm vi ngắn / (các khối dưới cùng) giống như GCN. Phương pháp này chứng minh hiệu suất thực nghiệm mạnh mẽ trên nhiều chuẩn mực tác vụ khác nhau, bao gồm dự đoán tính chất phân tử trong hóa học lượng tử & phân loại tài liệu trong mạng trích dẫn. Nó chỉ yêu cầu 1 số sửa đổi nhỏ đối với việc triển khai các GCN ban đầu. Tuy nhiên, nếu đồ thị đầu vào cực kỳ lớn (ví dụ: 1 số mạng xã hội lớn), thì bản thân thuật toán Lanczos sẽ là 1 nút thắt tính toán. Làm thế nào để cải thiện mô hình này trong bối cảnh vấn đề như vậy sẽ là 1 câu hỏi mở.

Here only introduce a few representative architectures of GNNs for node classification. There are also many other well-known architectures including gated GNNs (Li et al, 2016b) – which is mainly designed for output sequences – & GraphSAGE (Hamilton et al, 2017b) – which is mainly deigned for inductive setting of node classification.

– Ở đây chỉ giới thiệu 1 vài kiếntrúc GNN tiêu biểu cho phân loại nút. Ngoài ra còn có nhiều kiến trúc nổi tiếng khác, bao gồm GNN có cổng (Li & cộng sự, 2016b) – chủ yếu được thiết kế cho các chuỗi đầu ra – & GraphSAGE (Hamilton & cộng sự, 2017b) – chủ yếu được thiết kế cho việc thiết lập quy nạp phân loại nút.

- 4.3. Unsupervised GNNs. In this sect, review a few representative GNN-based methods for unsupervised learning on graph-structured data, including variational graph auto-encoders (Kipf & Welling, 2016) & deep graph infomax (Veličković et al, 2019).

  – GNN không giám sát. Trong phần này, hãy xem xét 1 số phương pháp tiêu biểu dựa trên GNN để học không giám sát trên dữ liệu có cấu trúc đồ thị, bao gồm bộ mã hóa tự động đồ thị biến thiên (Kipf & Welling, 2016) & deep graph infomax (Veličković et al, 2019).

  - 4.3.1. Variational Graph Auto-Encoders. Following variational auto-encoders (VAEs) (Kingma & Welling, 2014; Rezende et

al, 2014), variational graph auto-encoders (VGAEs) (Kipf & Welling, 2016) provide a framework for unsupervised learning on graph-structured data. In following, 1st review model & then discuss its advantages & disadvantages.
– Bộ mã hóa tự động đồ thị biến thiên. Tiếp theo là bộ mã hóa tự động đồ thị biến thiên (VAE) (Kingma & Welling, 2014; Rezende & cộng sự, 2014), bộ mã hóa tự động đồ thị biến thiên (VGAE) (Kipf & Welling, 2016) cung cấp 1 khuôn khổ cho học không giám sát trên dữ liệu có cấu trúc đồ thị. Trong phần tiếp theo, trước tiên hãy xem xét mô hình & sau đó thảo luận về ưu điểm & nhược điểm của nó.

· 4.3.1.1. Problem Setup. Suppose we are given an undirected graph $G = (V, E)$ with N nodes. Each node is associated with a node feature/attribute vector. Compactly denote all node features as a matrix $X \in \mathbb{R}^{N\times C}$. Adjacency matrix of graph is A. Assume self-loops are added to original graph G so that diagonal entries of A are 1. This is a convention in graph convolutional networks (GCNs) (Kipf & Welling, 2017b) & makes model consider a node's old representation while updating its new representation. Also assume each node is associated with a latent variable (collection of all latent variables is again compactly denoted as a matrix $Z \in \mathbb{R}^{N\times F}$). Interested in inferring latent variables of nodes in graph & decoding edges.
– Giả sử chúng ta được cho 1 đồ thị vô hướng $G = (V, E)$ với N nút. Mỗi nút được liên kết với 1 vectơ thuộc tính /nút. Ký hiệu gọn tất cả các đặc điểm của nút là 1 ma trận $X \in \mathbb{R}^{N\times C}$. Ma trận kề của đồ thị là A. Giả sử các vòng lặp tự thân được thêm vào đồ thị gốc G sao cho các mục đường chéo của A bằng 1. Đây là 1 quy ước trong mạng tích chập đồ thị (GCN) (Kipf & Welling, 2017b) & khiến mô hình xem xét biểu diễn cũ của 1 nút trong khi cập nhật biểu diễn mới của nó. Cũng giả sử mỗi nút được liên kết với 1 biến tiềm ẩn (tập hợp tất cả các biến tiềm ẩn 1 lần nữa được ký hiệu gọn là 1 ma trận $Z \in \mathbb{R}^{N\times F}$). Quan tâm đến việc suy ra các biến tiềm ẩn của các nút trong đồ thị & giải mã các cạnh.

· 4.3.1.2. Model. Similar to VAEs, VGAE model consists of an encoder $q_\phi(Z|A, X)$, a decoder $p_\theta(A|Z)$, & a prior $p(Z)$.
– Tương tự như VAE, mô hình VGAE bao gồm bộ mã hóa $q_\phi(Z|A, X)$, bộ giải mã $p_\theta(A|Z)$, & 1 bộ mã trước $p(Z)$.
Encoder. Goal of encoder: learn a distribution of latent variables associated with each node conditioning on node features X & adjacency matrix A. Could instantiate $q_\phi(Z|A, X)$ as a GNN where learnable parameters are ϕ. In particular, VGAE assumes an node-independent encoder as below,

$$\begin{aligned} q_\phi(Z|X, A) &= \prod_{i=1}^{N} q_\phi(\mathbf{z}_i|X, A), \\ q_\phi(\mathbf{z}_i|X, A) &= \mathcal{N}(\mathbf{z}_i|\mu_i, \text{diag}(\sigma_i^2)), \\ \boldsymbol{\mu}, \boldsymbol{\sigma} &= \text{GCN}_\phi(X, A), \end{aligned}$$

where $\mathbf{z}_i, \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i$ are ith rows of matrices $Z, \boldsymbol{\mu}, \boldsymbol{\sigma}$ resp. Basically, assume a multivariate Normal distribution with diagonal covariance as variational approximated distribution of latent vector per node (i.e., $\mathbf{z}_i$). Mean & diagonal covariance are predicted by encoder network, i.e., a GCN as described in Sect. 4.2.2. E.g., original paper uses a 2-layer GCN as follows:

$$\begin{aligned} \boldsymbol{\mu} &= \tilde{A}HW_{\boldsymbol{\mu}}, \\ \boldsymbol{\sigma} &= \tilde{A}WH_{\boldsymbol{\sigma}}, \\ H &= \text{ReLU}(\tilde{A}XW_0), \end{aligned}$$

where $\tilde{A} = D^{-1/2}AD^{-1/2}$ is symmetrically normalized adjacency matrix & D is degree matrix. Learnable parameters are thus $\phi = [W_{\boldsymbol{\mu}}, W_{\boldsymbol{\sigma}}, W_0]$.
– **Encoder.** Mục tiêu của bộ mã hóa: tìm hiểu phân phối các biến tiềm ẩn liên quan đến từng nút dựa trên đặc điểm nút X & ma trận kề A. Có thể khởi tạo $q_\phi(Z|A, X)$ dưới dạng GNN với các tham số có thể học được là ϕ. Cụ thể, VGAE giả định 1 bộ mã hóa độc lập với nút như sau,

$$\begin{aligned} q_\phi(Z|X, A) &= \prod_{i=1}^{N} q_\phi(\mathbf{z}_i|X, A), \\ q_\phi(\mathbf{z}_i|X, A) &= \mathcal{N}(\mathbf{z}_i|\mu_i, \text{diag}(\sigma_i^2)), \\ \boldsymbol{\mu}, \boldsymbol{\sigma} &= \text{GCN}_\phi(X, A), \end{aligned}$$

trong đó $\mathbf{z}_i, \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i$ là các hàng thứ i của ma trận $Z, \boldsymbol{\mu}, \boldsymbol{\sigma}$ tương ứng. Về cơ bản, giả sử phân phối chuẩn đa biến với hiệp phương sai đường chéo là phân phối xấp xỉ biến phân của vectơ tiềm ẩn trên mỗi nút (tức là $\mathbf{z}_i$). Hiệp phương sai trung bình & đường chéo được dự đoán bởi mạng mã hóa, tức là 1 GCN như được mô tả trong Mục 4.2.2. Ví dụ, bài báo gốc sử dụng GCN 2 lớp như sau:

$$\begin{aligned} \boldsymbol{\mu} &= \tilde{A}HW_{\boldsymbol{\mu}}, \\ \boldsymbol{\sigma} &= \tilde{A}WH_{\boldsymbol{\sigma}}, \\ H &= \text{ReLU}(\tilde{A}XW_0), \end{aligned}$$

trong đó $\tilde{A} = D^{-1/2}AD^{-1/2}$ là ma trận kề chuẩn hóa đối xứng & D là ma trận bậc. Do đó, các tham số có thể học được là $\phi = [W_{\boldsymbol{\mu}}, W_{\boldsymbol{\sigma}}, W_0]$.

Decoder. Given sampled latent variables, decoder aims at predicting connectivities among nodes. Original paper adopts a simple dot-product based predictor as below,

$$p(A|Z) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A_{ij}|\mathbf{z}_i, \mathbf{z}_j),$$
$$p(A_{ij}|\mathbf{z}_i, \mathbf{z}_j) = \boldsymbol{\sigma}(\mathbf{z}_i^\top \mathbf{z}_j),$$

where A_{ij} denotes (i, j)th element & $\boldsymbol{\sigma}(\cdot)$ is lgoistic sigmoid function. This decoder again assumes conditional independence among all possible edges for tractability. Note: there are no learnable parameters associated with this decoder. Only way to improve performance of decoder: learn good latent representations.
– **Decoder.** Với các biến tiềm ẩn được lấy mẫu, bộ giải mã hướng đến việc dự đoán khả năng kết nối giữa các nút. Bài báo gốc sử dụng 1 bộ dự đoán dựa trên tích vô hướng đơn giản như sau:

$$p(A|Z) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A_{ij}|\mathbf{z}_i, \mathbf{z}_j),$$
$$p(A_{ij}|\mathbf{z}_i, \mathbf{z}_j) = \boldsymbol{\sigma}(\mathbf{z}_i^\top \mathbf{z}_j),$$

trong đó A_{ij} biểu thị phần tử thứ (i, j) & $\boldsymbol{\sigma}(\cdot)$ là hàm sigmoid lgoistic. Bộ giải mã này 1 lần nữa giả định tính độc lập có điều kiện giữa tất cả các cạnh khả dĩ để đảm bảo tính dễ xử lý. Lưu ý: không có tham số nào có thể học được liên kết với bộ giải mã này. Cách duy nhất để cải thiện hiệu suất của bộ giải mã: học các biểu diễn tiềm ẩn tốt.
Prior. Prior distributions over latent variables are simply set to independent 0-mean Gaussians with unit variances, (4.39)

$$p(Z) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}).$$

This prior is fixed throughout learning as what typical VAEs do.
– **Prior.** Phân phối trước trên các biến tiềm ẩn chỉ đơn giản được đặt thành phân phối chuẩn Gauss độc lập có trung bình 0 với phương sai đơn vị, (4.39)

$$p(Z) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{z}_i|\mathbf{0}, \mathbf{I}).$$

Phân phối trước này được cố định trong suốt quá trình học như các VAE thông thường.
Objective & Learning. To learn encoder & decoder, one typically maximize evidence lower bound (ELBO) as in VAEs (4.40)

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi(Z|X,A)}[\log p(A|Z)] - \text{KL}(q_\theta(Z|X, A)||p(Z))$$

where $\text{KL}(q||p)$ is Kullback-Leibler divergence between distributions q, p. Note cannot directly maximize log likelihood since introduction of latent variables Z includes a high-dimensional integral which is intractable. Instead maximize ELBO in (4.40) which is a lower bound of log likelihood. However, 1st expectation term is again intractable. One often resorts to Monte Carlo estimation by sampling a few Z from encoder $q_\theta(Z|X, A)$ & evaluating term using samples. To maximize objective, one can perform stochastic gradient descent along with reparameterization trick (Kingma & Welling, 2014). Note: reparametrization trick is necessary since need to back-propagate through sampling in aforementioned Monte Carlo estimation term to compute gradient w.r.t. parameters of encoder.
– **Mục tiêu & Học tập.** Để học bộ mã hóa & giải mã, người ta thường tối đa hóa giới hạn dưới của bằng chứng (ELBO) như trong VAE (4.40)

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_\phi(Z|X,A)}[\log p(A|Z)] - \text{KL}(q_\theta(Z|X, A)||p(Z))$$

trong đó $\text{KL}(q||p)$ là phân kỳ Kullback-Leibler giữa các phân phối q, p. Lưu ý không thể tối đa hóa trực tiếp log likelihood vì việc đưa vào các biến tiềm ẩn Z bao gồm 1 tích phân nhiều chiều không thể giải được. Thay vào đó, hãy tối đa hóa ELBO trong (4.40) là giới hạn dưới của log likelihood. Tuy nhiên, số hạng kỳ vọng thứ nhất cũng không thể giải được. Người ta thường dùng đến ước lượng Monte Carlo bằng cách lấy mẫu 1 vài Z từ bộ mã hóa $q_\theta(Z|X, A)$ & đánh giá số hạng bằng mẫu. Để tối đa hóa mục tiêu, người ta có thể thực hiện giảm gradient ngẫu nhiên kết hợp với thủ thuật tham số hóa lại (Kingma & Welling, 2014). Lưu ý: thủ thuật tham số hóa lại là cần thiết vì cần phải lan truyền ngược qua việc lấy mẫu trong số hạng ước lượng Monte Carlo đã đề cập ở trên để tính toán gradient theo các tham số của bộ mã hóa.

· 4.3.1.3. Discussion. VGAE model is popular in literature mainly due to its simplicity & good empirical performances. E.g., since there are no learnable parameters for prior & decoder, model is quite light-weight & learning process is fast. Moreover, VGAE model is versatile in way that once we learned a good encoder, i.e., good latent representations, can use them for predicting edges (link prediction), node attributes, & so on. On other side, VGAE model is still limited in following ways. 1st, it cannot serve as a good generative model for graphs as what VAEs do for images since decoder is not learnable. One could simply design some learnable decoder. However, not clear: goal of learning good latent representations & generating graphs with good qualities are always well-aligned. More exploration along this direction

would be fruitful. 2nd, independence assumption is exploited for both encoder & decoder which might be very limited. More structural dependence (e.g., auto-regressive) would be desirable to improve model capacity. 3rd, as discussed in original paper, prior may be potentially a poor choice. At last, for link prediction in practice, one may need to add weighting of edges vs. non-edges in decoder term & carefully tune it since graphs may be very sparse.

– Mô hình VGAE phổ biến trong các tài liệu chủ yếu nhờ tính đơn giản & hiệu suất thực nghiệm tốt. Ví dụ, vì không có tham số học được cho bộ giải mã trước, nên mô hình khá nhẹ & quá trình học diễn ra nhanh. Hơn nữa, mô hình VGAE rất linh hoạt, 1 khi chúng ta đã học được 1 bộ mã hóa tốt, tức là các biểu diễn tiềm ẩn tốt, có thể sử dụng chúng để dự đoán các cạnh (dự đoán liên kết), thuộc tính nút, & vân vân. Mặt khác, mô hình VGAE vẫn còn hạn chế ở những điểm sau. Thứ nhất, nó không thể đóng vai trò là 1 mô hình sinh tốt cho đồ thị như những gì VAE làm cho hình ảnh vì bộ giải mã không thể học được. Người ta có thể chỉ cần thiết kế 1 bộ giải mã có thể học được. Tuy nhiên, vẫn chưa rõ ràng: mục tiêu học các biểu diễn tiềm ẩn tốt & tạo ra đồ thị với chất lượng tốt luôn được sắp xếp hợp lý. Việc khám phá thêm theo hướng này sẽ rất hiệu quả. Thứ hai, giả định độc lập được khai thác cho cả bộ mã hóa & bộ giải mã, điều này có thể rất hạn chế. Sự phụ thuộc về mặt cấu trúc hơn (ví dụ: tự hồi quy) sẽ là mong muốn để cải thiện khả năng của mô hình. Thứ ba, như đã thảo luận trong bài báo gốc, phương pháp trước có thể là 1 lựa chọn kém hiệu quả. Cuối cùng, để dự đoán liên kết trong thực tế, có thể cần thêm trọng số của các cạnh so với các cạnh không phải cạnh trong thuật ngữ giải mã & điều chỉnh cẩn thận vì đồ thị có thể rất thưa thớt.

∗ 4.3.2. Deep Graph Infomax. Following Mutual Information Neural Estimation (MINE) (Belghazi et al, 2018) & Deep Infomax (Hjelm et al, 2018), Deep Graph Infomax (Veličković et al, 2019) is unsupervised learning framework that learns graph representations via principle of mutual information maximization.

– Deep Graph Infomax. Theo Ước tính thông tin tương hỗ (MINE) (Belghazi & cộng sự, 2018) & Deep Infomax (Hjelm & cộng sự, 2018), Deep Graph Infomax (Veličković & cộng sự, 2019) là khuôn khổ học tập không giám sát học các biểu diễn đồ thị thông qua nguyên tắc tối đa hóa thông tin tương hỗ.

· 4.3.2.1. Problem Setup. Following original paper, explain model under single-graph setup, i.e., node feature matrix X & graph adjacency matrix A of a single graph are provided as input. Extensions to other problem setups like transductive & inductive learning settings will be discussed in Sect. 4.3.2.3. Goal: learn node representations in an unsupervised way. After node representations are learned, one can apply some simple linear (logistic regression) classifier on top of representations to perform supervised tasks like node classification.

– Tiếp theo bài báo gốc, hãy giải thích mô hình trong thiết lập đồ thị đơn, tức là ma trận đặc trưng nút X & ma trận kề đồ thị A của 1 đồ thị đơn được cung cấp làm đầu vào. Việc mở rộng các thiết lập bài toán khác như thiết lập học chuyển đổi & quy nạp sẽ được thảo luận trong Phần 4.3.2.3. Mục tiêu: học biểu diễn nút theo cách không giám sát. Sau khi học được biểu diễn nút, người ta có thể áp dụng 1 số bộ phân loại tuyến tính đơn giản (hồi quy logistic) lên trên các biểu diễn để thực hiện các tác vụ có giám sát như phân loại nút.

∗ 4.3.2.2. Model. Main idea of model: maximize local mutual information between a node representation (capturing local graph information) & graph representation (capturing global graph information). By doing so, learned node representation should capture global graph information as much as possible. Let us denote graph encoder as ε which could be any GNN discussed before, e.g., a 2-layer GCN. Can obtain all node representations as $H = \varepsilon(X, A)$ where representation $\mathbf{h}_i$ of any node i should contain some local information near node i. Specifically, k-layer GCN should be able to leverage node information that is k-hop away. To get global graph information, one could use a readout layer/function to process all node representations, i.e., $\mathbf{s} = \mathcal{R}(H)$, where readout function $\mathcal{R}$ could be some learnable pooling function or simply an average operator.

– Ý tưởng chính của mô hình: tối đa hóa thông tin tương hỗ cục bộ giữa biểu diễn nút (ghi lại thông tin đồ thị cục bộ) & biểu diễn đồ thị (ghi lại thông tin đồ thị toàn cục). Bằng cách làm như vậy, biểu diễn nút đã học sẽ ghi lại thông tin đồ thị toàn cục càng nhiều càng tốt. Hãy ký hiệu bộ mã hóa đồ thị là ε có thể là bất kỳ GNN nào đã thảo luận trước đó, ví dụ: GCN 2 lớp. Có thể thu được tất cả các biểu diễn nút là $H = \varepsilon(X, A)$ trong đó biểu diễn $\mathbf{h}_i$ của bất kỳ nút i nào sẽ chứa 1 số thông tin cục bộ gần nút i. Cụ thể, GCN lớp k sẽ có thể tận dụng thông tin nút cách k-hop. Để có được thông tin đồ thị toàn cục, người ta có thể sử dụng hàm lớp đọc để xử lý tất cả các biểu diễn nút, tức là $\mathbf{s} = \mathcal{R}(H)$, trong đó hàm đọc $\mathcal{R}$ có thể là 1 số hàm gộp có thể học được hoặc chỉ đơn giản là 1 toán tử trung bình.

Objective. Given local node representation $\mathbf{h}_i$ & global graph representation $\mathbf{s}$, natural next step is to compute their mutual information. Recall definition of mutual information is as follows, (4.11)

$$\mathrm{MI}(\mathbf{h}, \mathbf{s}) = \iint p(\mathbf{h}, \mathbf{s}) \log \frac{p(\mathbf{h}, \mathbf{s})}{p(\mathbf{h}) p(\mathbf{s})} \, \mathrm{d}\mathbf{h} \, \mathrm{d}\mathbf{s}.$$

However, maximizing local mutual information alone is not enough to learn useful representations as shown in (Hjelm et al, 2018). To develop a more practical objective, authors in (Veličković et al, 2019) instead use a noise-contrastive type objective following Deep Infomax (Hjelm et al, 2018), (4.42)

$$\mathcal{L} = \frac{1}{N+M} \left(\sum_{i=1}^{N} \mathbb{E}_{(X,A)} [\log \mathcal{D}(\mathbf{h}_i, \mathbf{s})] + \sum_{j=1}^{M} \mathbb{E}_{(\tilde{X}, \tilde{A})} [\log(1 - \mathcal{D}(\tilde{\mathbf{h}}_j, \mathbf{s}))] \right),$$

where $\mathcal{D}$ is a binary classifier which takes both node representation $\mathbf{h}_i$ & graph representation $\mathbf{s}$ as input & predicts whether pair $(\mathbf{h}_i, \mathbf{s})$ comes from joint distribution $p(\mathbf{h}, \mathbf{s})$ (positive class) or product of marginals $p(\mathbf{h}_i)p(\mathbf{s})$ (negative class). Denote $\tilde{\mathbf{h}}_j$ as jth node representation from negative sample. Numbers of positive & negative samples are N, M,

resp. Explain how to draw positive & negative samples shortly. Overall objective is thus negative binary cross-entropy for training a probabilistic classifier. Note: this objective is same type of distance as used in generative adversarial networks (GANs) (Goodfellow et al, 2014b) which is shown to be proportional to Jensen-Shannon divergence (Goodfellow et al, 2014b; Nowozin et al, 2016). As verified by (Hjelm et al, 2018), maximizing Jensen-Shannon divergence based mutual information estimator behaves similarly (i.e., they have an approximately monotonic relationship) to directly maximizing mutual information. Therefore, maximizing objective in (4.42) is expected to maximize mutual information. Moreover, freedom of choosing negative samples makes method more likely to learn useful representations than maximizing vanilla mutual information.

– **Mục tiêu.** Với biểu diễn nút cục bộ $\mathbf{h}_i$ & biểu diễn đồ thị toàn cục $\mathbf{s}$, bước tiếp theo tự nhiên là tính toán thông tin tương hỗ của chúng. Nhắc lại định nghĩa về thông tin tương hỗ như sau, (4.11)

$$\mathrm{MI}(\mathbf{h}, \mathbf{s}) = \iint p(\mathbf{h}, \mathbf{s}) \log \frac{p(\mathbf{h}, \mathbf{s})}{p(\mathbf{h})p(\mathbf{s})} \, \mathrm{d}\mathbf{h} \, \mathrm{d}\mathbf{s}.$$

Tuy nhiên, chỉ riêng việc tối đa hóa thông tin tương hỗ cục bộ là không đủ để học các biểu diễn hữu ích như được trình bày trong (Hjelm & cộng sự, 2018). Để phát triển 1 mục tiêu thực tế hơn, các tác giả trong (Veličković & cộng sự, 2019) thay vào đó sử dụng 1 mục tiêu loại tương phản nhiễu theo Deep Infomax (Hjelm & cộng sự, 2018), (4.42)

$$\mathcal{L} = \frac{1}{N+M} \left(\sum_{i=1}^{N} \mathbb{E}_{(X,A)}[\log \mathcal{D}(\mathbf{h}_i, \mathbf{s})] + \sum_{j=1}^{M} \mathbb{E}_{(\tilde{X},\tilde{A})}[\log(1 - \mathcal{D}(\tilde{\mathbf{h}}_j, \mathbf{s}))] \right),$$

trong đó $\mathcal{D}$ là 1 bộ phân loại nhị phân sử dụng cả hai biểu diễn nút **BiudinØthh**$_i$ & $\mathbf{s}$ làm đầu vào & dự đoán liệu cặp $(\mathbf{h}_i, \mathbf{s})$ đến từ phân phối chung $p(\mathbf{h}, \mathbf{s})$ (lớp dương) hay tích của các biên $p(\mathbf{h}_i)p(\mathbf{s})$ (lớp âm). Ký hiệu $\tilde{\mathbf{h}}_j$ là biểu diễn nút thứ j từ mẫu âm. Số lượng mẫu dương & âm tương ứng là N, M. Giải thích ngắn gọn cách vẽ mẫu dương & âm. Do đó, mục tiêu chung là entropy chéo nhị phân âm để huấn luyện bộ phân loại xác suất. Lưu ý: mục tiêu này cùng loại với khoảng cách được sử dụng trong mạng đối kháng sinh sinh (GAN) (Goodfellow & cộng sự, 2014b), được chứng minh là tỷ lệ thuận với độ phân kỳ Jensen-Shannon (Goodfellow & cộng sự, 2014b; Nowozin & cộng sự, 2016). Như đã được xác minh bởi (Hjelm & cộng sự, 2018), việc tối đa hóa ước lượng thông tin tương hỗ dựa trên độ phân kỳ Jensen-Shannon hoạt động tương tự (tức là chúng có mối quan hệ gần đơn điệu) với việc tối đa hóa trực tiếp thông tin tương hỗ. Do đó, việc tối đa hóa mục tiêu trong (4.42) được kỳ vọng sẽ tối đa hóa thông tin tương hỗ. Hơn nữa, việc tự do lựa chọn các mẫu âm khiến phương pháp có nhiều khả năng học được các biểu diễn hữu ích hơn so với việc tối đa hóa thông tin tương hỗ vani.

Negative Sampling. To generate positive samples, one can directly sample a few nodes from graph to construct pairs $(\mathbf{h}_i, \mathbf{s})$. For negative samples, one can generate them via corrupting original graph data, denoting as $(\tilde{X}, \tilde{A}) = \mathcal{C}(X, A)$. In practice, one can choose various forms of this corruption function $\mathcal{C}$. E.g., authors in (Veličković et al, 2019) suggest to keep adjacency matrix to be same & corrupt node feature X by row-wise shuffling. Other possibilities of corruption function include randomly sampling subgraphs & applying Dropout (Srivastava et al, 2014) to node features.

– **Lấy mẫu âm.** Để tạo ra các mẫu dương, người ta có thể lấy mẫu trực tiếp 1 vài nút từ đồ thị để xây dựng các cặp $(\mathbf{h}_i, \mathbf{s})$. Đối với các mẫu âm, người ta có thể tạo chúng bằng cách làm hỏng dữ liệu đồ thị gốc, ký hiệu là $(\tilde{X}, \tilde{A}) = \mathcal{C}(X, A)$. Trên thực tế, người ta có thể chọn nhiều dạng khác nhau của hàm làm hỏng này $\mathcal{C}$. Ví dụ: các tác giả trong (Veličković et al, 2019) đề xuất giữ nguyên ma trận kề & làm hỏng đặc trưng nút X bằng cách xáo trộn từng hàng. Các khả năng khác của hàm làm hỏng bao gồm lấy mẫu ngẫu nhiên các đồ thị con & áp dụng Dropout (Srivastava et al, 2014) cho các đặc trưng nút.

Once positive & negative samples were collected, one can learn representations via maximizing objective in (4.42). Summarize training process of Deep Graph Infomax as follows:

1. Sample negative examples via corruption function $(\tilde{X}, \tilde{A}) \sim \mathcal{C}(X, A)$.
2. Compute node representations of positive samples $H = \{\mathbf{h}_1, \ldots, \mathbf{h}_N\} = \varepsilon(X, A)$.
3. Compute node representations of negative samples $\tilde{H} = \{\tilde{\mathbf{h}}_1, \ldots, \tilde{\mathbf{h}}_M\} = \varepsilon(\tilde{X}, \tilde{A})$.
4. Compute graph representation via readout function $\mathbf{s} = \mathcal{R}(H)$.
5. Update parameters of $\varepsilon, \mathcal{D}, \mathcal{R}$ via gradient ascent to maximize (4.42).

– Sau khi thu thập được các mẫu dương & âm, ta có thể học các biểu diễn thông qua việc tối đa hóa mục tiêu trong (4.42). Tóm tắt quy trình huấn luyện của Deep Graph Infomax như sau:

1. Lấy mẫu các ví dụ âm thông qua hàm hỏng $(\tilde{X}, \tilde{A}) \sim \mathcal{C}(X, A)$.
2. Tính toán các biểu diễn nút của các mẫu dương $H = \{\mathbf{h}_1, \ldots, \mathbf{h}_N\} = \varepsilon(X, A)$.
3. Tính toán các biểu diễn nút của các mẫu âm $\tilde{H} = \{\tilde{\mathbf{h}}_1, \ldots, \tilde{\mathbf{h}}_M\} = \varepsilon(\tilde{X}, \tilde{A})$.
4. Tính toán biểu diễn đồ thị thông qua hàm đọc $\mathbf{s} = \mathcal{R}(H)$.
5. Cập nhật các tham số của $\varepsilon, \mathcal{D}, \mathcal{R}$ thông qua phép tăng dần gradient để tối đa hóa (4.42).

∗ 4.3.2.3. Discussion. Deep Graph Infomax is an efficient unsupervised representation learning method for graph-structured data. Implementation of encoder, readout, binary cross-entropy type of loss are all straightforward. Mini-batch training does not necessarily need to store whole graph since readout can be applied to a set of subgraphs as well. Therefore, method is memory-efficient. Also, processing of positive & negative samples can be done in parallel. Moreover, authors proved: minimizing cross-entropy type of classification error can be used ot maximize mutual information under certain conditions,

e.g., readout function is injective & input feature comes from a finite set. However, choice of corruption function seems to be crucial to ensure satisfying empirical performances. There seems no such a universally good corruption function. One needs to do trial-&-error to obtain a proper one depending on task/dataset.

– Deep Graph Infomax là 1 phương pháp học biểu diễn không giám sát hiệu quả cho dữ liệu có cấu trúc đồ thị. Việc triển khai bộ mã hóa, đọc kết quả, loại mất mát entropy chéo nhị phân đều rất đơn giản. Huấn luyện lô nhỏ không nhất thiết phải lưu trữ toàn bộ đồ thị vì đọc kết quả cũng có thể được áp dụng cho 1 tập hợp các đồ thị con. Do đó, phương pháp này tiết kiệm bộ nhớ. Ngoài ra, việc xử lý các mẫu dương & âm có thể được thực hiện song song. Hơn nữa, các tác giả đã chứng minh: có thể sử dụng phương pháp giảm thiểu loại lỗi phân loại entropy chéo để tối đa hóa thông tin tương hỗ trong 1 số điều kiện nhất định, ví dụ: hàm đọc kết quả là hàm đơn ánh & đặc trưng đầu vào đến từ 1 tập hữu hạn. Tuy nhiên, việc lựa chọn hàm hỏng dường như rất quan trọng để đảm bảo hiệu suất thực nghiệm thỏa mãn. Dường như không có 1 hàm hỏng nào tốt như vậy trên mọi phương diện. Người ta cần phải thử nghiệm để có được 1 hàm hỏng phù hợp tùy thuộc vào tập dữ liệu tác vụ.

- 4.4. Oversmoothing Problem. Training deep neural networks by stacking multiple layers of GNNs usually yields inferior results, which is a common problem observed in many different GNN architectures. This is mainly due to problem of over-smoothing, which is 1st explicitly studied in (Li et al, 2018b). (Li et al, 2018b) showed: graph convolutional network (Kipf & Welling, 2017b) is a special case of Laplacian smoothing: (4.43)

$$Y = (1 - \gamma I)X + \gamma \tilde{A}_{rw} X,$$

where $\tilde{A}_{rw} = \tilde{D}^{-1}\tilde{A}$, which defines transitional probabilities between nodes on graphs. GCN corresponds to a special case of Laplacian smoothing with $\gamma = 1$ & symmetric matrix $\tilde{A}_{\text{sym}} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is used. Laplacian smoothing will push nodes belonging to same clusters to take similar representations, which are beneficial for downstream tasks e.g. node classification. However, when GCNs go deep, node representations suffer from problem of over-smoothing, i.e., all nodes will have similar representations. As a result, performance on downstream tasks suffer as well. This phenomenon has later been pointed out by a few other later work as well such as (Zhao & Akoglu, 2019; Li et al, 2018b; Xu et al, 2018a; Li et al, 2019c; Rong et al, 2020b).

– Vấn đề làm mịn quá mức. Việc huấn luyện mạng nơ-ron sâu bằng cách xếp chồng nhiều lớp GNN thường mang lại kết quả kém hơn, đây là 1 vấn đề phổ biến được quan sát thấy trong nhiều kiến trúc GNN khác nhau. Điều này chủ yếu là do vấn đề làm mịn quá mức, được nghiên cứu rõ ràng lần đầu tiên trong (Li & cộng sự, 2018b). (Li & cộng sự, 2018b) đã chỉ ra: mạng tích chập đồ thị (Kipf & Welling, 2017b) là 1 trường hợp đặc biệt của làm mịn Laplacian: (4.43)

$$Y = (1 - \gamma I)X + \gamma \tilde{A}_{rw} X,$$

trong đó $\tilde{A}_{rw} = \tilde{D}^{-1}\tilde{A}$, xác định xác suất chuyển tiếp giữa các nút trên đồ thị. GCN tương ứng với 1 trường hợp đặc biệt của làm mịn Laplacian với $\gamma = 1$ & ma trận đối xứng $\tilde{A}_{\text{sym}} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ được sử dụng. Làm mịn Laplacian sẽ đẩy các nút thuộc cùng 1 cụm để lấy các biểu diễn tương tự, điều này có lợi cho các tác vụ hạ lưu, ví dụ như phân loại nút. Tuy nhiên, khi GCN đi sâu, các biểu diễn nút gặp phải vấn đề làm mịn quá mức, tức là tất cả các nút sẽ có các biểu diễn tương tự. Kết quả là, hiệu suất của các tác vụ hạ lưu cũng bị ảnh hưởng. Hiện tượng này sau đó cũng đã được chỉ ra bởi 1 số công trình khác sau này như (Zhao & Akoglu, 2019; Li & cộng sự, 2018b; Xu & cộng sự, 2018a; Li & cộng sự, 2019c; Rong & cộng sự, 2020b).

PairNorm. (Zhao & Akoglu, 2019). Next, present a method called PairNorm for alleviating problem of oversmoothing when GNNs go deep. Essential idea of PairNorm: keep total pairwise squared distance (TPSD) of node representations unchanged, which is same as that of original node feature X. Let $\tilde{H}$ be output of node representations by graph convolution, which will be input of PairNorm, & $\hat{H}$ is output of PairNorm. Goal of PairNorm: normalize $\tilde{H}$ s.t. after normalization $\text{TPSD}(\hat{H}) = \text{TPSD}(X)$. I.e., (4.44)

$$\sum_{(i,j)\in\mathcal{E}} \|\hat{H}_i - \hat{H}_j\|^2 + \sum_{(i,j)\notin\mathcal{E}} \|\hat{H}_i - \hat{H}_j\|^2 = \sum_{(i,j)\in\mathcal{E}} \|X_i - X_j\|^2 + \sum_{(i,j)\notin\mathcal{E}} \|X_i - X_j\|^2.$$

– **PairNorm.** (Zhao & Akoglu, 2019). Tiếp theo, trình bày 1 phương pháp gọi là PairNorm để giảm thiểu vấn đề làm mịn quá mức khi GNN đi sâu. Ý tưởng cốt lõi của PairNorm: giữ nguyên tổng khoảng cách bình phương từng cặp (TPSD) của các biểu diễn nút, giống như khoảng cách bình phương từng cặp (TPSD) của đặc trưng nút gốc X. Giả sử $\tilde{H}$ là đầu ra của các biểu diễn nút bằng phép tích chập đồ thị, sẽ là đầu vào của PairNorm, & $\hat{H}$ là đầu ra của PairNorm. Mục tiêu của PairNorm: chuẩn hóa $\tilde{H}$ sau khi chuẩn hóa $\text{TPSD}(\hat{H}) = \text{TPSD}(X)$. Tức là, (4.44)

$$\sum_{(i,j)\in\mathcal{E}} \|\hat{H}_i - \hat{H}_j\|^2 + \sum_{(i,j)\notin\mathcal{E}} \|\hat{H}_i - \hat{H}_j\|^2 = \sum_{(i,j)\in\mathcal{E}} \|X_i - X_j\|^2 + \sum_{(i,j)\notin\mathcal{E}} \|X_i - X_j\|^2.$$

[... pp. 60–61]

- 4.5. Summary. In this chap, give a comprehensive introduction to different architectures of GNNs for node classification. These neural networks can be generally classified into 2 categories including supervised & unsupervised approaches. For supervised approaches, main difference among different architectures lie in how to propagate messages between nodes, how to aggregate messages from neighbors, & how to combine aggregated messages from neighbors with node representation itself. For unsupervised approaches, main difference comes from designing objective function. Also discuss a common problem of

training deep neural networks – oversmoothing, & introduce a method to tackle it. In future, promising directions on GNNs include theoretical analysis for understanding behaviors of GNns, & applying them to a variety of fields & domains e.g. recommender systems, knowledge graphs, drug & material discovery, computer vision, & natural language understanding.

– Trong chương này, chúng tôi sẽ giới thiệu toàn diện về các kiến trúc khác nhau của GNN để phân loại nút. Các mạng nơ-ron này thường có thể được phân loại thành 2 loại bao gồm các phương pháp có giám sát & không giám sát. Đối với các phương pháp có giám sát, sự khác biệt chính giữa các kiến trúc khác nhau nằm ở cách truyền tin nhắn giữa các nút, cách tổng hợp tin nhắn từ các nút lân cận, & cách kết hợp các tin nhắn tổng hợp từ các nút lân cận với chính biểu diễn nút. Đối với các phương pháp không giám sát, sự khác biệt chính đến từ việc thiết kế hàm mục tiêu. Đồng thời thảo luận về 1 vấn đề phổ biến khi huấn luyện mạng nơ-ron sâu – làm mịn quá mức, & giới thiệu 1 phương pháp để giải quyết vấn đề này. Trong tương lai, các hướng triển vọng về GNN bao gồm phân tích lý thuyết để hiểu hành vi của GNn, & áp dụng chúng vào nhiều lĩnh vực & miền khác nhau, ví dụ như hệ thống đề xuất, đồ thị tri thức, khám phá thuốc & vật liệu, thị giác máy tính, & hiểu ngôn ngữ tự nhiên.

Node classification task is 1 of most important tasks in GNNs. Node representation learning techniques introduce in this chap are corner stone for all other tasks for rest of book, including graph classification task (Chap. 9), link prediction (Chap. 10), graph generation task (Chap. 11), & so on. Familiar with learning methodologies & design principles of node representation learning is key to deeply understanding other fundamental research directions like Theoretical analysis (Chap. 5), Scalability (Chap. 6), Explainability (Chap. 7), & Adversarial Robustness (Chap. 8).

– Nhiệm vụ phân loại nút là 1 trong những nhiệm vụ quan trọng nhất trong mạng nơ-ron nhân tạo (GNN). Các kỹ thuật học biểu diễn nút được giới thiệu trong chương này là nền tảng cho tất cả các nhiệm vụ khác trong phần còn lại của sách, bao gồm nhiệm vụ phân loại đồ thị (Chương 9), dự đoán liên kết (Chương 10), nhiệm vụ tạo đồ thị (Chương 11), v.v. Việc làm quen với các phương pháp học tập & nguyên tắc thiết kế của học biểu diễn nút là chìa khóa để hiểu sâu sắc các hướng nghiên cứu cơ bản khác như Phân tích lý thuyết (Chương 5), Khả năng mở rộng (Chương 6), Khả năng giải thích (Chương 7), & Tính mạnh mẽ đối nghịch (Chương 8).

- 5. Expressive Power of GNNs. Success of neural networks is based on their strong expressive power that allows them to approximate complex nonlinear mappings from features to predictions. Since universal approximation theorem by (Cybenko, 1989), many studies have proved: feedforward neural networks can approximate any function of interest. However, these results have not been applied to GNNs due to inductive bias imposed by additional constraints on GNN parameter space. New theoretical studies are needed to better understand these constraints & characterize expressive power of GNNs.

  – Sức mạnh biểu đạt của GNN. Sự thành công của mạng nơ-ron dựa trên sức mạnh biểu đạt mạnh mẽ của chúng, cho phép chúng xấp xỉ các ánh xạ phi tuyến tính phức tạp từ đặc trưng đến dự đoán. Kể từ định lý xấp xỉ phổ quát của (Cybenko, 1989), nhiều nghiên cứu đã chứng minh: mạng nơ-ron truyền thẳng có thể xấp xỉ bất kỳ hàm nào mong muốn. Tuy nhiên, những kết quả này chưa được áp dụng cho GNN do sai số quy nạp do các ràng buộc bổ sung trên không gian tham số GNN. Cần có các nghiên cứu lý thuyết mới để hiểu rõ hơn về những ràng buộc này & mô tả sức mạnh biểu đạt của GNN.

  In this chap, review recent progress on expressive power of GNNs in graph representation learning. Start by introducing most widely-used GNN framework – message passing – & analyze its power & limitations. Next introduce some recently proposed techniques to overcome these limitations, e.g. injecting random attributes, injecting deterministic distance attributes, & building higher-order GNNs. Present key insights of these techniques & highlight their advantages & disadvantages.

  – Trong chương này, chúng ta sẽ xem xét những tiến bộ gần đây về khả năng biểu đạt của GNN trong học biểu diễn đồ thị. Bắt đầu bằng việc giới thiệu khuôn khổ GNN được sử dụng rộng rãi nhất – truyền thông điệp – & phân tích sức mạnh & hạn chế của nó. Tiếp theo, chúng ta sẽ giới thiệu 1 số kỹ thuật được đề xuất gần đây để khắc phục những hạn chế này, ví dụ như chèn thuộc tính ngẫu nhiên, chèn thuộc tính khoảng cách xác định, & xây dựng GNN bậc cao. Trình bày những hiểu biết chính về các kỹ thuật này & làm nổi bật ưu điểm & nhược điểm của chúng.

  - 5.1. Introduction. ML problems can be abstracted as learning a mapping f^* from some feature space to some target space. Solution to this problem is typically given by a model f_θ that intends to approximate f^* via optimizing some parameter θ. In practice, ground truth f^* is a priori typically unknown. Therefore, one may expect model f_θ to approximate a rather broad range of f^*. An estimate of how broad such a range could be, called model's *expressive power*, provides an important measure of model potential. Desirable to have models with a more expressive power that may learn more complex mapping functions.

    – Các bài toán học máy có thể được trừu tượng hóa như việc học 1 phép ánh xạ f^* từ 1 không gian đặc trưng nào đó đến 1 không gian đích nào đó. Giải pháp cho bài toán này thường được đưa ra bởi 1 mô hình f_θ, mô hình này dự định xấp xỉ f^* thông qua việc tối ưu hóa 1 tham số θ nào đó. Trên thực tế, giá trị thực tế f^* thường là 1 ẩn số. Do đó, người ta có thể kỳ vọng mô hình f_θ xấp xỉ 1 phạm vi khá rộng của f^*. Ước tính về phạm vi rộng như vậy, được gọi là *sức mạnh biểu đạt* của mô hình, cung cấp 1 thước đo quan trọng về tiềm năng của mô hình. Mong muốn có các mô hình với sức mạnh biểu đạt cao hơn để có thể học các hàm ánh xạ phức tạp hơn.

    Neural networks (NNs) are well known for their great expressive power. Specifically, Cybenko (1989) 1st proved: any continuous function defined over a compact space could be uniformly approximated by neural networks with sigmoid activation functions & only 1 hidden layer. Later, this result got generalized to any squashing activation functions by (Hornik et al, 1989).

    – Mạng nơ-ron (NN) nổi tiếng với khả năng biểu đạt tuyệt vời. Cụ thể, Cybenko (1989) lần đầu tiên chứng minh: bất kỳ hàm liên tục nào được xác định trên 1 không gian compact đều có thể được xấp xỉ đồng đều bởi mạng nơ-ron với hàm kích

hoạt sigmoid & chỉ 1 lớp ẩn. Sau đó, kết quả này được tổng quát hóa cho bất kỳ hàm kích hoạt nén nào bởi (Hornik & cộng sự, 1989).

However, these seminal findings are insufficient to explain current unprecedented success of NNs in practice because their strong expressive power only demonstrates: model f_θ is able to approximate f^* but does not guarantee: model obtained via training $\hat{f}$ indeed approximates f^*. Fig. 5.1: Amount of Data vs. Performance of different models. illustrates a well-known curve of Amount of Data vs. Performance of ML models (Ng, 2011). NN-based methods may only outperform traditional methods given sufficient data. 1 important reason: NNs as ML models are still governed by fundamental tradeoff between data amount & model complexity Fig. 5.2: Training & testing errors with & without inductive bias can dramatically affect expressive power of models. Although NNs could be rather expressive, they are likely to overfit training examples when paired with more parameters. Therefore, necessary for practice to build NNs that can maintain strong expressive power while constraints are imposed on their parameters. At same time, a good theoretical understanding of expressive power of NNs with constraints on their parameters is needed.

– Tuy nhiên, những phát hiện quan trọng này là không đủ để giải thích thành công chưa từng có hiện nay của NN trong thực tế vì sức mạnh biểu đạt mạnh mẽ của chúng chỉ chứng minh: mô hình f_θ có thể xấp xỉ f^* nhưng không đảm bảo: mô hình thu được thông qua đào tạo $\hat{f}$ thực sự xấp xỉ f^*. Hình 5.1: Lượng dữ liệu so với Hiệu suất của các mô hình khác nhau. minh họa đường cong nổi tiếng về Lượng dữ liệu so với Hiệu suất của các mô hình ML (Ng, 2011). Các phương pháp dựa trên NN chỉ có thể vượt trội hơn các phương pháp truyền thống khi có đủ dữ liệu. 1 lý do quan trọng: NN là mô hình ML vẫn bị chi phối bởi sự đánh đổi cơ bản giữa lượng dữ liệu & độ phức tạp của mô hình Hình 5.2: Lỗi đào tạo & kiểm tra với & không có độ lệch quy nạp có thể ảnh hưởng đáng kể đến sức mạnh biểu đạt của các mô hình. Mặc dù NN có thể khá biểu đạt, nhưng chúng có khả năng quá khớp với các ví dụ đào tạo khi được ghép nối với nhiều tham số hơn. Do đó, cần thiết cho thực hành để xây dựng các NN có thể duy trì sức mạnh biểu đạt mạnh mẽ trong khi các ràng buộc được áp đặt cho các tham số của chúng. Đồng thời, cần có sự hiểu biết lý thuyết tốt về sức mạnh biểu đạt của NN với các ràng buộc về tham số của chúng.

In practice, constraints on parameters are typically obtained from our prior knowledge of data; these are referred to as inductive biases. Some significant results about expressive power of NNs with inductive bias have been shown recently. Yarotsky (2017); Liang & Srikant (2017) have proved: deep neural networks (DNNs), by stacking multiple hidden layers, can achieve good enough approximation with significantly fewer parameters than shallow NNs. Architecture of DNNs leverages fact: data has typically a hierarchical structure. DNNs are agnostic to type of data, while dedicated neural network architectures have been developed to support specific types of data. RNNs (Hochreiter & Schmidhuber, 1997) or convolution neural networks (CNNs) (LeCun et al, 1989) were proposed to process time series & images, resp. In these 2 types of data, effective patterns typically hold translation invariance in time & in space, resp. To match this invariance, RNNs & CNNs adopt inductive bias that their parameters have shared across time & space Fig. 5.3: Illustration of 1D translation invariance/variance. RNNs/CNNs use translation invariance to share parameters. Parameter-sharing mechanism works as a constraint on parameters & limits expressive power of RNNs & CNNs. However, RNNs & CNNs have been shown to have sufficient expressive power to learn translation invariant functions (Siegelmann & Sontag, 1995; Cohen & Shashua, 2016; Khrulkov et al, 2018), which leads to great practical success of RNNs & CNNs in processing time series & images.

– Trong thực tế, các ràng buộc về tham số thường thu được từ kiến thức trước đó của chúng ta về dữ liệu; những ràng buộc này được gọi là độ lệch quy nạp. 1 số kết quả quan trọng về sức mạnh biểu đạt của NN với độ lệch quy nạp đã được trình bày gần đây. Yarotsky (2017); Liang & Srikant (2017) đã chứng minh: mạng nơ-ron sâu (DNN), bằng cách xếp chồng nhiều lớp ẩn, có thể đạt được phép xấp xỉ đủ tốt với số lượng tham số ít hơn đáng kể so với NN nông. Kiến trúc của DNN tận dụng thực tế: dữ liệu thường có cấu trúc phân cấp. DNN không phụ thuộc vào loại dữ liệu, trong khi kiến trúc mạng nơ-ron chuyên dụng đã được phát triển để hỗ trợ các loại dữ liệu cụ thể. RNN (Hochreiter & Schmidhuber, 1997) hoặc mạng nơ-ron tích chập (CNN) (LeCun & cộng sự, 1989) đã được đề xuất để xử lý chuỗi thời gian & hình ảnh, tương ứng. Trong 2 loại dữ liệu này, các mẫu hiệu quả thường giữ tính bất biến tịnh tiến theo thời gian & trong không gian, tương ứng. Để phù hợp với tính bất biến này, RNN & CNN áp dụng độ lệch quy nạp mà các tham số của chúng chia sẻ theo thời gian & không gian Hình 5.3: Minh họa về tính bất biến tịnh tiến 1D/. RNN/CNN sử dụng tính bất biến tịnh tiến để chia sẻ tham số. Cơ chế chia sẻ tham số hoạt động như 1 ràng buộc đối với các tham số & giới hạn khả năng biểu đạt của RNN & CNN. Tuy nhiên, RNN & CNN đã được chứng minh là có đủ khả năng biểu đạt để học các hàm bất biến tịnh tiến (Siegelmann & Sontag, 1995; Cohen & Shashua, 2016; Khrulkov & cộng sự, 2018), điều này dẫn đến thành công thực tế to lớn của RNN & CNN trong việc xử lý hình ảnh chuỗi thời gian.

Recently, many studies have focused on a new type of NNs, termed graph neural networks (GNNs) (Scarselli et al, 2008; Bruna et al, 2014; Kipf & Welling, 2017a; Bronstein et al, 2017; Gilmer et al, 2017; Hamilton et al, 2017b; Battaglia et al, 2018). These aim to capture inductive bias of graphs/networks, another important type of data. Graphs are commonly used to model complex relations & interactions between multiple elements & have been widely used in ML applications, e.g. community detection, recommendation systems, molecule property prediction, & medicine design (Fortunato, 2010; Fouss et al, 2007; Pires et al, 2015). Compared to time series & images, which are well-structured & represented by tables or grids, graphs are irregular & thus introduce new challenges. A fundamental assumption behind ML on graphs: targets for prediction should be invariant to order of nodes of graph. To match this assumption, GNNs hold a general inductive bias termed permutation invariance. In particular, output given by GNNs should be independent of how node indices of a graph are assigned & thus in which order are they processed. GNNs require their parameters to be independent from node ordering & are shared across entire graph Fig. 5.4: This illustrates how GNNs are designed to maintain permutation invariance. Because of this new parameter sharing mechanism in GNNs, new theoretical tools are needed to characterize their expressive power.

– Gần đây, nhiều nghiên cứu đã tập trung vào 1 loại NN mới, được gọi là mạng nơ-ron đồ thị (GNN) (Scarselli & cộng sự,

2008; Bruna & cộng sự, 2014; Kipf & Welling, 2017a; Bronstein & cộng sự, 2017; Gilmer & cộng sự, 2017; Hamilton & cộng sự, 2017b; Battaglia & cộng sự, 2018). Những nghiên cứu này nhằm mục đích nắm bắt độ lệch quy nạp của đồ thị/mạng, 1 loại dữ liệu quan trọng khác. Đồ thị thường được sử dụng để mô hình hóa các mối quan hệ phức tạp & tương tác giữa nhiều yếu tố & đã được sử dụng rộng rãi trong các ứng dụng ML, ví dụ: phát hiện cộng đồng, hệ thống đề xuất, dự đoán tính chất phân tử, & thiết kế thuốc (Fortunato, 2010; Fouss & cộng sự, 2007; Pires & cộng sự, 2015). So với chuỗi thời gian & hình ảnh, được cấu trúc tốt & biểu diễn bằng bảng hoặc lưới, đồ thị không đều & do đó đưa ra những thách thức mới. 1 giả định cơ bản đằng sau ML trên đồ thị: các mục tiêu dự đoán phải bất biến theo thứ tự các nút của đồ thị. Để phù hợp với giả định này, GNN có 1 độ lệch quy nạp chung được gọi là bất biến hoán vị. Cụ thể, đầu ra do GNN cung cấp phải độc lập với cách chỉ số nút của đồ thị được gán & do đó chúng được xử lý theo thứ tự nào. GNN yêu cầu các tham số của chúng độc lập với thứ tự nút & được chia sẻ trên toàn bộ đồ thị Hình 5.4: Điều này minh họa cách GNN được thiết kế để duy trì bất biến hoán vị. Do cơ chế chia sẻ tham số mới này trong GNN, cần có các công cụ lý thuyết mới để mô tả sức mạnh biểu đạt của chúng.

Analyzing expressive power of GNNs is challenging, as this problem is closely related to some long-standing problems in graph theory. To understand this connection, consider following example of how a GNN would predict whether a graph structure corresponds to a valid molecule. GNN should be able to identify whether this graph structure is same, similar, or very different from graph structures that are known to correspond to valid molecules. Measuring whether 2 graphs have same structure involves addressing graph isomorphism problem, in which no P solutions have yet been found (Helfgott et al, 2017). In addition, measuring whether 2 graphs have a similar structure requires contending with graph edit distance problem, which is even harder to address than graph isomorphism problem Lewis et al, 1983).

– Phân tích sức mạnh biểu đạt của GNN là 1 thách thức, vì vấn đề này liên quan chặt chẽ đến 1 số vấn đề tồn tại lâu đời trong lý thuyết đồ thị. Để hiểu mối liên hệ này, hãy xem xét ví dụ sau về cách GNN dự đoán liệu cấu trúc đồ thị có tương ứng với 1 phân tử hợp lệ hay không. GNN phải có khả năng xác định liệu cấu trúc đồ thị này có giống, tương tự hay rất khác so với các cấu trúc đồ thị đã biết là tương ứng với các phân tử hợp lệ hay không. Việc đo lường xem 2 đồ thị có cùng cấu trúc hay không liên quan đến việc giải quyết vấn đề đồng cấu đồ thị, trong đó chưa tìm thấy giải pháp P nào (Helfgott & cộng sự, 2017). Ngoài ra, việc đo lường xem 2 đồ thị có cấu trúc tương tự nhau hay không đòi hỏi phải đối mặt với vấn đề khoảng cách chỉnh sửa đồ thị, thậm chí còn khó giải quyết hơn vấn đề đồng cấu đồ thị (Lewis & cộng sự, 1983).

Great progress has been made recently on characterizing expressive power of GNNs, especially on how to match their power with traditional graph algorithms & how to build more powerful GNNs that overcome limitation of those algorithms. Delve more into these recent efforts further along in this chap. In particular, compared to previous introductions (Hamilton, 2020; Sato, 2020), focus on recent key insights & techniques that yield more powerful GNNs. Specifically, introduce standard message-passing GNNs that are able to achieve limit of 1D Weisfeiler-Lehman test (Weisfeiler & Leman, 1968), a widely-used algorithm to test for graph isomorphism. Also discuss a number of strategies to overcome limitations of Weisfeiler-Lehman test – including attaching random attributes, attaching deterministic distance attributes, & leveraging higher-order structures.

– Gần đây đã có những tiến bộ lớn trong việc mô tả sức mạnh biểu đạt của GNN, đặc biệt là về cách kết hợp sức mạnh của chúng với các thuật toán đồ thị truyền thống & cách xây dựng các GNN mạnh hơn để khắc phục hạn chế của các thuật toán đó. Hãy đi sâu hơn vào những nỗ lực gần đây này trong phần sau của chương này. Đặc biệt, so với các phần giới thiệu trước đây (Hamilton, 2020; Sato, 2020), hãy tập trung vào những hiểu biết quan trọng gần đây & các kỹ thuật tạo ra các GNN mạnh hơn. Cụ thể, hãy giới thiệu các GNN truyền thông điệp tiêu chuẩn có thể đạt được giới hạn của phép kiểm tra Weisfeiler-Lehman 1D (Weisfeiler & Leman, 1968), 1 thuật toán được sử dụng rộng rãi để kiểm tra tính đồng cấu của đồ thị. Cũng thảo luận về 1 số chiến lược để khắc phục hạn chế của phép kiểm tra Weisfeiler-Lehman – bao gồm việc gắn các thuộc tính ngẫu nhiên, gắn các thuộc tính khoảng cách xác định, & tận dụng các cấu trúc bậc cao hơn.

In Sect. 5.2, formulate graph representation learning problems that GNNs target. In Sect. 5.3, review most widely used GNN framework, message passing neural network, describing limitations of its expressive power & discussing its efficient implementations. In Sect. 5.4, introduce a number of methods that make GNNs more powerful than message passing neural network. In Sect. 5.5, conclude this chap by discussing further research directions.

– Trong Phần 5.2, xây dựng các bài toán học biểu diễn đồ thị mà GNN hướng đến. Trong Phần 5.3, xem xét khuôn khổ GNN được sử dụng rộng rãi nhất, mạng nơ-ron truyền thông điệp, mô tả những hạn chế về khả năng biểu đạt của nó & thảo luận về các triển khai hiệu quả. Trong Phần 5.4, giới thiệu 1 số phương pháp giúp GNN mạnh hơn mạng nơ-ron truyền thông điệp. Trong Phần 5.5, kết thúc chương này bằng cách thảo luận về các hướng nghiên cứu tiếp theo.

Fig. 5.5: An illustration of expressive power of NNs & GNNs & their affects on performance of learned models. (a) ML problems aim to learn mapping from feature space to target space based on several observed examples. (b) Expressive power of NNs refer to gap between 2 spaces $\mathcal{F}, \hat{\mathcal{F}}'$. Although NNs are expressive ($\hat{\mathcal{F}}'$ is dense in $\mathcal{F}$), learned model f' based on NNs may differ significantly from f^* due to their overfit of limited observed data. (c) Suppose f^* is known to be permutation invariant, as it works on graph-structured data. Then, space of potential mapping functions is reduced from $\mathcal{F}'$ to a much smaller space $\mathcal{F}$ that only includes permutation invariant functions. If adopt GNNs, space of mapping functions that can be approximated simultaneously reduces to $\hat{\mathcal{F}}$. Gap between $\mathcal{F}, \hat{\mathcal{F}}$ characterizes expressive power of GNNs. (d) Although GNNs are less expressive than general NNs ($\hat{\mathcal{F}} \subset \hat{\mathcal{F}}'$), learned model based on GNNs f is a much better approximation of f^* as opposed to the one based on NNs $\hat{f}'$. Therefore, for graph-structured data, our understanding of expressive power of GNNs, i.e., gap between $\mathcal{F}, \hat{\mathcal{F}}$, is much more relevant than that of NNs.

– Hình 5.5: Minh họa về sức mạnh biểu đạt của NN & GNN & ảnh hưởng của chúng đến hiệu suất của các mô hình đã học. (a) Các bài toán ML nhằm mục đích học ánh xạ từ không gian đặc trưng đến không gian mục tiêu dựa trên 1 số ví dụ đã quan sát. (b) Sức mạnh biểu đạt của NN đề cập đến khoảng cách giữa 2 không gian $\mathcal{F}, \hat{\mathcal{F}}'$. Mặc dù NN có tính biểu đạt ($\hat{\mathcal{F}}'$ dày đặc trong

$\mathcal{F}$), mô hình đã học f' dựa trên NN có thể khác đáng kể so với f^* do quá phù hợp với dữ liệu quan sát hạn chế. (c) Giả sử f^* được biết là bất biến hoán vị, vì nó hoạt động trên dữ liệu có cấu trúc đồ thị. Khi đó, không gian của các hàm ánh xạ thế được giảm từ $\mathcal{F}'$ thành 1 không gian nhỏ hơn nhiều $\mathcal{F}$ chỉ bao gồm các hàm bất biến hoán vị. Nếu áp dụng GNN, không gian các hàm ánh xạ có thể được xấp xỉ đồng thời sẽ giảm xuống còn $\hat{\mathcal{F}}$. Khoảng cách giữa $\mathcal{F}, \hat{\mathcal{F}}$ đặc trưng cho sức mạnh biểu đạt của GNN. (d) Mặc dù GNN ít biểu đạt hơn các NN tổng quát ($\hat{\mathcal{F}} \subset \hat{\mathcal{F}}'$), mô hình học dựa trên GNN f là phép xấp xỉ tốt hơn nhiều của f^* so với mô hình dựa trên NN $\hat{f}'$. Do đó, đối với dữ liệu có cấu trúc đồ thị, hiểu biết của chúng ta về sức mạnh biểu đạt của GNN, tức là khoảng cách giữa $\mathcal{F}, \hat{\mathcal{F}}$, có liên quan hơn nhiều so với hiểu biết của NN.

- 5.2. Graph Representation Learning & Problem Formulation. In this sect, set up formal definition of graph representation learning problems, their fundamental assumption, & their inductive bias. Also discuss relationships between different notions of graph representation learning problems frequently studied in recent literature. 1st, start by defining graph-structured data.

– Học Biểu diễn Đồ thị & Xây dựng Bài toán. Trong phần này, hãy thiết lập định nghĩa chính thức về các bài toán học biểu diễn đồ thị, giả định cơ bản của chúng, & độ lệch quy nạp của chúng. Đồng thời thảo luận về mối quan hệ giữa các khái niệm khác nhau về các bài toán học biểu diễn đồ thị thường được nghiên cứu trong các tài liệu gần đây. 1. Trước tiên, hãy bắt đầu bằng cách định nghĩa dữ liệu có cấu trúc đồ thị.

Definition 1 (Graph-structured data). *Let $G = (V, E, X)$ denote an attributed graph, where V is node set, E is edge set, & $X \in \mathbb{R}^{|V| \times F}$ are node attributes. Each row of $X, X_v \in \mathbb{R}^F$ refers to attributes on node $v \in V$. In practice, graphs are usually sparse, i.e., $|E| \ll |V|^2$. Introduce $A \in \{0,1\}^{|V| \times |V|}$ to denote adjacency matrix of G s.t. $A_{uv} = 1$ iff $(u, v) \in E$. Combining adjacency matrix & node attributes, may also denote $G = (A, X)$. Moreover, if G is unattributed with no node attributes, can assume: all elements in X are constant. Later, also use $V[G]$ to denote entire node set of a particular graph G.*

Định nghĩa 1 (Dữ liệu có cấu trúc đồ thị). *Cho $G = (V, E, X)$ biểu thị 1 đồ thị được gán, trong đó V là tập nút, E là tập cạnh, & $X \in \mathbb{R}^{|V| \times F}$ là các thuộc tính nút. Mỗi hàng của $X, X_v \in \mathbb{R}^F$ tham chiếu đến các thuộc tính trên nút $v \in V$. Trong thực tế, đồ thị thường thưa thớt, tức là $|E| \ll |V|^2$. Giới thiệu $A \in \{0,1\}^{|V| \times |V|}$ để biểu thị ma trận kề của G khi $A_{uv} = 1$ khi & chỉ khi $(u, v) \in E$. Kết hợp ma trận kề & các thuộc tính nút, cũng có thể biểu thị $G = (A, X)$. Hơn nữa, nếu G không được gán & không có thuộc tính nút nào, có thể giả sử: tất cả các phần tử trong X là hằng số. Sau đó, cũng sử dụng $V[G]$ để biểu thị toàn bộ tập hợp nút của 1 đồ thị cụ thể G.*

Goal of graph representation learning: learn a model by taking graph-structured data as input & then mapping it so that certain prediction targets are matched. Different graph representation learning problems may apply to a varying number of nodes in a graph. E.g., for node classification, a prediction is made for each node, for each link/relation prediction on a pair of nodes, & for each graph classification or graph property prediction on entire node set V. Can unify all these problems as graph representation learning.

– Mục tiêu của học biểu diễn đồ thị: học 1 mô hình bằng cách lấy dữ liệu có cấu trúc đồ thị làm đầu vào & sau đó ánh xạ nó sao cho các mục tiêu dự đoán nhất định được khớp. Các bài toán học biểu diễn đồ thị khác nhau có thể áp dụng cho số lượng nút khác nhau trong đồ thị. Ví dụ: đối với phân loại nút, 1 dự đoán được đưa ra cho mỗi nút, cho mỗi dự đoán quan hệ liên kết trên 1 cặp nút, & cho mỗi phân loại đồ thị hoặc dự đoán thuộc tính đồ thị trên toàn bộ tập nút V. Có thể thống nhất tất cả các bài toán này thành học biểu diễn đồ thị.

Definition 2 (Graph representation learning GRL). *Feature space is defined as $X \coloneqq \Gamma \times P$, where Γ: space of graph-structured data & P includes all node subsets of interest, given a graph $G \in \Gamma$. Then, a point in X can be denoted as (G, S), where S is a subset of nodes that are in G. Later, call (G, S) as a graph representation learning (GRL) example. Each GRL example $(G, S) \in X$ is associated with a target y in target space Y. Suppose ground-truth association function between features & targets is denoted by $f^* : X \to Y$, $f^*(G, S) = y$. Given a set of training examples $\Xi = \{(G^{(i)}, S^{(i)}, y^{(i)})\}_{i=1}^k$ & a set of testing examples $\Psi = \{(\tilde{G}^{(i)}, \tilde{S}^{(i)}, \tilde{y}^{(i)})\}_{i=1}^k$, a* graph representation learning *problem is to learn a function f based on Ξ s.t. f is close to f^* to Ψ.*

Định nghĩa 2. *Không gian đặc trưng được định nghĩa là $X \coloneqq \Gamma \times P$, trong đó Γ: không gian dữ liệu có cấu trúc đồ thị & P bao gồm tất cả các tập con nút quan tâm, cho trước 1 đồ thị $G \in \Gamma$. Khi đó, 1 điểm trong X có thể được ký hiệu là (G, S), trong đó S là 1 tập con của các nút nằm trong G. Sau đó, hãy gọi (G, S) là 1 ví dụ học biểu diễn đồ thị (GRL). Mỗi ví dụ GRL $(G, S) \in X$ được liên kết với 1 mục tiêu y trong không gian mục tiêu Y. Giả sử hàm liên kết thực tế giữa các đặc trưng & mục tiêu được ký hiệu là $f^* : X \to Y$, $f^*(G, S) = y$. Với 1 tập hợp các ví dụ đào tạo $\Xi = \{(G^{(i)}, S^{(i)}, y^{(i)})\}_{i=1}^k$ & 1 tập hợp các ví dụ kiểm tra $\Psi = \{(\tilde{G}^{(i)}, \tilde{S}^{(i)}, \tilde{y}^{(i)})\}_{i=1}^k$, 1 bài toán* biểu diễn đồ thị *là học 1 hàm f dựa trên Ξ s. f gần với f^* của Ψ.*

Above definition is general in sense that in a GRL example $(G, S) \in X$, G provides both raw & structural features on which some prediction for a node subset S of interest is to be made. Below, further list a few frequently-investigated learning problems that may be formulated as graph representation learning problems.

– Định nghĩa trên mang tính tổng quát theo nghĩa là trong ví dụ GRL $(G, S) \in X$, G cung cấp cả các đặc điểm thô & cấu trúc mà dựa vào đó có thể đưa ra dự đoán cho tập con nút S cần quan tâm. Dưới đây, chúng tôi xin liệt kê thêm 1 số bài toán học tập thường được nghiên cứu, có thể được xây dựng dưới dạng các bài toán học tập biểu diễn đồ thị.

Remark 4 (Graph classification problem/Graph-level prediction – Bài toán phân loại đồ thị/Dự đoán cấp độ đồ thị). *Node set S of interest is entire node set $V[G]$ by default. Space of graph-structured data Γ typically contains multiple graphs. Target space Y contains labels of different graphs. Later, for graph-level prediction, use G to denote a GRL example instead of (G, S) for notational simplicity.*

– Tập nút S được quan tâm mặc định là toàn bộ tập nút $V[G]$. Không gian dữ liệu có cấu trúc đồ thị Γ thường chứa nhiều đồ thị. Không gian mục tiêu Y chứa nhãn của các đồ thị khác nhau. Sau này, để dự đoán ở cấp độ đồ thị, hãy sử dụng G để biểu thị 1 ví dụ GRL thay vì (G, S) để đơn giản hóa ký hiệu.

Remark 5 (Node classification problem/Node-level prediction – Vấn đề phân loại nút/Dự đoán cấp độ nút)**.** *In a GRL example (G, S), S corresponds to 1 single node of interest. Corresponding G can be defined in different ways. On 1 hand, only nodes close to S provide effective features. In this case, G may be set as induced local subgraph around S. Different G's for different S's may come from a single graph. On other hand, 2 nodes that are far apart on 1 graph still hold mutual impact & can be used as a feature to make a prediction on another graph. In that case, G needs to include a large portion of a graph or even entire graph.*

– Trong ví dụ GRL (G, S), S tương ứng với 1 nút đơn cần quan tâm. G tương ứng có thể được định nghĩa theo nhiều cách khác nhau. 1 mặt, chỉ những nút gần S mới cung cấp các đặc trưng hiệu quả. Trong trường hợp này, G có thể được đặt là đồ thị con cục bộ cảm ứng xung quanh S. Các G khác nhau cho các S khác nhau có thể đến từ 1 đồ thị duy nhất. Mặt khác, 2 nút cách xa nhau trên cùng 1 đồ thị vẫn có tác động lẫn nhau & có thể được sử dụng làm đặc trưng để đưa ra dự đoán trên 1 đồ thị khác. Trong trường hợp đó, G cần bao gồm 1 phần lớn đồ thị hoặc thậm chí toàn bộ đồ thị.

Remark 6 (Link prediction problem/Node-pair-level prediction – Vấn đề dự đoán liên kết/Dự đoán cấp độ cặp nút)**.** *In a GRL example (G, S), S corresponds to a pair of nodes of interest. Similar to node classification problem, G for each example may be an induced subgraph around S or entire graph. Target space Y contains 0-1 labels that indicate whether there is a probable link between 2 nodes. Y may also be generalized to include labels that reflect types of links to be predicted.*

– Trong ví dụ GRL (G, S), S tương ứng với 1 cặp nút cần quan tâm. Tương tự như bài toán phân loại nút, G cho mỗi ví dụ có thể là 1 đồ thị con cảm ứng xung quanh S hoặc toàn bộ đồ thị. Không gian mục tiêu Y chứa các nhãn 0-1 cho biết liệu có tồn tại liên kết khả dĩ giữa 2 nút hay không. Y cũng có thể được khái quát hóa để bao gồm các nhãn phản ánh loại liên kết cần dự đoán.

Introduce fundamental assumption used in most graph representation learning problems.

– Giới thiệu giả định cơ bản được sử dụng trong hầu hết các bài toán học biểu diễn đồ thị.

Definition 3 (Isomorphism)**.** *Consider 2 GRL examples $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)}) \in X$. Suppose $G^{(1)} = (A^{(1)}, X^{(1)}), G^{(2)} = (A^{(2)}, X^{(2)})$. If there exists a bibjective mapping $\pi : V[G^{(1)}] \to V[G^{(2)}]$, $i \in [2]$, s.t. $A^{(1)}_{uv} = A^{(2)}_{\pi(u)\pi(v)}, X^{(1)}_u = X^{(2)}_{\pi(u)}$ & π also gives a bijective mapping between $S^{(1)}, S^{(2)}$, call $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)})$ are isomorphic, denoted as $(G^{(1)}, S^{(1)}) \cong (G^{(2)}, S^{(2)})$. When particular bijective mapping π should be highlighted, use notation $(G^{(1)}, S^{(1)}) \overset{\pi}{\cong} (G^{(2)}, S^{(2)})$. If there is no such a π, call they are non-isomorphic, denoted as $(G^{(1)}, S^{(1)}) \ncong (G^{(2)}, S^{(2)})$.*

– Xét 2 ví dụ GRL $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)}) \in X$. Giả sử $G^{(1)} = (A^{(1)}, X^{(1)}), G^{(2)} = (A^{(2)}, X^{(2)})$. Nếu tồn tại 1 ánh xạ nhị phân $\pi : V[G^{(1)}] \to V[G^{(2)}]$, $i \in [2]$, s.t. $A^{(1)}_{uv} = A^{(2)}_{\pi(u)\pi(v)}, X^{(1)}_u = X^{(2)}_{\pi(u)}$ & π cũng đưa ra 1 ánh xạ song ánh giữa $S^{(1)}, S^{(2)}$, gọi $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)})$ là đồng cấu, ký hiệu là $(G^{(1)}, S^{(1)}) \cong (G^{(2)}, S^{(2)})$. Khi cần làm nổi bật 1 ánh xạ song ánh cụ thể π, hãy sử dụng ký hiệu $(G^{(1)}, S^{(1)}) \overset{\pi}{\cong} (G^{(2)}, S^{(2)})$. Nếu không có π nào như vậy, hãy gọi chúng là không đẳng cấu, được ký hiệu là $(G^{(1)}, S^{(1)}) \ncong (G^{(2)}, S^{(2)})$.

Assumption 1 (Fundamental assumption in graph representation learning – Giả định cơ bản trong học biểu diễn đồ thị)**.** *Consider a graph representation learning problem with a feature space X & its corresponding target space Y. Pick any 2 GRL examples $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)}) \in X$. Fundamental assumption says that if $(G^{(1)}, S^{(1)}) \cong (G^{(2)}, S^{(2)})$, their corresponding targets in Y are same.*

– Xét 1 bài toán học biểu diễn đồ thị với không gian đặc trưng X & không gian đích tương ứng Y. Chọn bất kỳ 2 ví dụ GRL $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)}) \in X$. Giả định cơ bản nói rằng nếu $(G^{(1)}, S^{(1)}) \cong (G^{(2)}, S^{(2)})$, thì các đích tương ứng của chúng trong Y là giống nhau.

Due to this fundamental assumption, natural to introduce corresponding *permutation invariance* as inductive bias that all models of graph representation learning should satisfy.

– Do giả định cơ bản này, việc đưa ra *bất biến hoán vị* tương ứng như độ lệch quy nạp mà tất cả các mô hình học biểu diễn đồ thị phải đáp ứng là điều tự nhiên.

Definition 4 (Permutation invariance)**.** *A model f satisfies permutation invariance if for any $(G^{(1)}, S^{(1)}) \cong (G^{(2)}, S^{(2)})$, $f(G^{(1)}, S^{(1)}) = f(G^{(2)}, S^{(2)})$.*

– Mô hình f thỏa mãn tính bất biến hoán vị nếu với bất kỳ $(G^{(1)}, S^{(1)}) \cong (G^{(2)}, S^{(2)})$, $f(G^{(1)}, S^{(1)}) = f(G^{(2)}, S^{(2)})$.

Now may define expressive power of a model for graph representation learning problems.

– Bây giờ có thể xác định sức mạnh biểu đạt của mô hình cho các vấn đề học tập về biểu diễn đồ thị.

Definition 5 (Expressive power – Sức mạnh biểu đạt)**.** *Consider a feature space X of a graph representation learning problem & a model f defined on X. Define another space $X(f)$ as a subspace of quotient space $X/\cong$ s.t. for 2 GRL examples $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)}) \in X(f)$, $f(G^{(1)}, S^{(1)}) \neq f(G^{(2)}, S^{(2)})$. Then, size of $X(f)$ characterizes expressive power of f. For 2 models, $f^{(1)}, f^{(2)}$, if $X(f^{(1)}) \supset X(f^{(2)})$, say that $f^{(1)}$ is more expressive than $f^{(2)}$.*

– Xét 1 không gian đặc trưng X của 1 bài toán học biểu diễn đồ thị & 1 mô hình f được xác định trên X. Định nghĩa 1 không gian $X(f)$ khác là 1 không gian con của không gian thương $X/\cong$ s.t. cho 2 ví dụ GRL $(G^{(1)}, S^{(1)}), (G^{(2)}, S^{(2)}) \in X(f)$, $f(G^{(1)}, S^{(1)}) \neq f(G^{(2)}, S^{(2)})$. Khi đó, kích thước của $X(f)$ đặc trưng cho sức mạnh biểu đạt của f. Với 2 mô hình, $f^{(1)}, f^{(2)}$, nếu $X(f^{(1)}) \supset X(f^{(2)})$, hãy nói rằng $f^{(1)}$ biểu đạt hơn $f^{(2)}$.

Remark 7. *Note expressive power in Def. 5.5, characterized by how a model can distinguish non-isomorphic GRL examples, does not exactly match traditional expressive power used for NNs in sense of functional approximation. Actually, Def. 5.5 is strictly weaker because distinguishing any non-isomorphic GRL examples does not necessarily indicate that we can approximate any function f^* defined over X. However, if a model f cannot distinguish 2 non-isomorphic features, f is definitely unable to approximate function f^* that maps these 2 examples to 2 different targets. Some recent studies have been able to prove some equivalence between distinguishing non-isomorphic features & permutation invariant function approximations under weak assumptions & applying involved techniques (Chen et al, 2019f; Azizian & Lelarge, 2020).*

– Lưu ý sức mạnh biểu đạt trong Định nghĩa 5.5, được đặc trưng bởi cách 1 mô hình có thể phân biệt các ví dụ GRL không đẳng cấu, không hoàn toàn khớp với sức mạnh biểu đạt truyền thống được sử dụng cho các mạng nơ-ron nhân tạo theo nghĩa xấp xỉ hàm. Trên thực tế, Định nghĩa 5.5 yếu hơn nhiều vì việc phân biệt bất kỳ ví dụ GRL không đẳng cấu nào không nhất thiết chỉ ra rằng chúng ta có thể xấp xỉ bất kỳ hàm f^* nào được xác định trên X. Tuy nhiên, nếu 1 mô hình f không thể phân biệt 2 đặc điểm không đẳng cấu, thì f chắc chắn không thể xấp xỉ hàm f^* ánh xạ 2 ví dụ này tới 2 mục tiêu khác nhau. 1 số nghiên cứu gần đây đã có thể chứng minh 1 số tính tương đương giữa việc phân biệt các đặc điểm không đẳng cấu & xấp xỉ hàm bất biến hoán vị theo các giả định yếu & áp dụng các kỹ thuật liên quan (Chen et al, 2019f; Azizian & Lelarge, 2020).

Trivial to provide expressive power of a model f for graph representation learning if f does not satisfy permutation invariance. Without such a constraint, NNs can approximate all continuous functions (Cybenko, 1989), which include continuous functions that distinguish any non-isomorphic GRL examples. Therefore, key question discussed in this chap: "How to build most expressive permutation invariant models, GNNs in particular, for graph representation learning problems?"

– Việc cung cấp sức mạnh biểu đạt của mô hình f cho việc học biểu diễn đồ thị là không cần thiết nếu f không thỏa mãn tính bất biến hoán vị. Nếu không có ràng buộc này, mạng nơ-ron nhân tạo (NN) có thể xấp xỉ tất cả các hàm liên tục (Cybenko, 1989), bao gồm các hàm liên tục phân biệt bất kỳ ví dụ GRL nào không đẳng cấu. Do đó, câu hỏi chính được thảo luận trong chương này: "Làm thế nào để xây dựng các mô hình bất biến hoán vị biểu đạt nhất, đặc biệt là GNN, cho các bài toán học biểu diễn đồ thị?"

- 5.3. Power of Message Passing GNNs.
  - 5.3.1. Preliminaries: Neural Networks for Sets. Start by reviewing NNs with sets (multisets) as their input, since a set can be viewed as a simplified-version of a graph where all edges are removed. By definition, order of elements of a set does not impact output; models that encode sets naturally provide an important building block for encoding graphs. Term this approach invariant pooling.

    – Sơ bộ: Mạng nơ-ron cho tập hợp. Hãy bắt đầu bằng việc xem xét các mạng nơ-ron nhân tạo với tập hợp (đa tập hợp) làm đầu vào, vì 1 tập hợp có thể được xem như 1 phiên bản đơn giản hóa của 1 đồ thị trong đó tất cả các cạnh đều bị loại bỏ. Theo định nghĩa, thứ tự các phần tử của 1 tập hợp không ảnh hưởng đến đầu ra; các mô hình mã hóa tập hợp tự nhiên cung cấp 1 nền tảng quan trọng cho việc mã hóa đồ thị. Hãy gọi cách tiếp cận này là gộp bất biến.

    Definition 6 (Multiset)**.** *A* multiset *is a set where its elements can be repetitive, i.e., they are present multiple times. In this chap, assume by default that all sets are multisets & thus allow repetitive elements. In situations where this is not case, will indicate otherwise.*

    – 1 đa tập *là 1 tập hợp mà các phần tử của nó có thể lặp lại, tức là chúng xuất hiện nhiều lần. Trong chương này, mặc định giả định rằng tất cả các tập hợp đều là đa tập & do đó cho phép các phần tử lặp lại. Trong trường hợp không phải vậy, sẽ chỉ ra trường hợp ngược lại.*

    Definition 7 (Invariant pooling – Gộp bất biến)**.** *Given a multiset of vectors $S = \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$ where $\mathbf{a}_i \in \mathbb{R}^F$ & F is an arbitrary constant, an invariant pooling refers to a mapping, denoted as $q(S)$, that is invariant to order of elements in S.*
    – Với 1 tập hợp đa vectơ $S = \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$ trong đó $\mathbf{a}_i \in \mathbb{R}^F$ & F là hằng số tùy ý, 1 nhóm bất biến đề cập đến 1 ánh xạ, được ký hiệu là $q(S)$, bất biến theo thứ tự các phần tử trong S.

    Some widely-used invariant pooling operations include: sum pooling $q(S) = \sum_{i=1}^k \mathbf{a}_i$, mean pooling $q(S) = \frac{1}{k}\sum_{i=1}^k \mathbf{a}_i$, & max pooling $[q(S)]_j = \max_{i\in[F]} a_{ij}$, $\forall j \in [F]$. Zaheer et al (2017) show: any invariant poolings of a set S can be approximated by $q(S) = \phi\left(\sum_{i=1}^k \psi(\mathbf{a}_i)\right)$, where ϕ, ψ are functions that may be approximated by fully connected NNs, provided that $\mathbf{a}_i$, $i \in [k]$ comes from a countable universe.This statement can be generalized to the case where S is a multiset (Xu et al, 2019d).

    – 1 số phép toán gộp bất biến được sử dụng rộng rãi bao gồm: gộp tổng $q(S) = \sum_{i=1}^k \mathbf{a}_i$, gộp trung bình $q(S) = \frac{1}{k}\sum_{i=1}^k \mathbf{a}_i$, & gộp tối đa $[q(S)]_j = \max_{i\in[F]} a_{ij}$, $\forall j \in [F]$. Zaheer et al (2017) chỉ ra: bất kỳ nhóm bất biến nào của 1 tập hợp S đều có thể được xấp xỉ bằng $q(S) = \phi\left(\sum_{i=1}^k \psi(\mathbf{a}_i)\right)$, trong đó ϕ, ψ là các hàm có thể được xấp xỉ bằng các mạng nơron kết nối đầy đủ, với điều kiện là $\mathbf{a}_i$, $i \in [k]$ xuất phát từ 1 vũ trụ đếm được. Tuyên bố này có thể được khái quát hóa cho trường hợp S là 1 tập hợp đa (Xu et al, 2019d).
  - 5.3.2. Message Passing GNNs. Message passing is most widely-used framework to build GNNs (Gilmer et al, 2017). Given a graph $G = (V, E, X)$, message passing framework encodes each node $v \in V$ with a vector representation $\mathbf{h}_v$ & keeps updating this node representation by iteratively collecting representations of its neighbors & applying neural network layers to perform a nonlinear transformation of those collections:
    1. Initialize node vector representations as node attributes: $\mathbf{h}_v^{(0)} \leftarrow X_v$, $\forall v \in V$.
    2. Update each node representation based on message passing over graph structure. In lth layer, $l \in [L]$, perform following steps:

· Message: $\mathbf{m}_{vu}^{(1)} \leftarrow \text{MSG}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)})$, $\forall (u,v) \in E$

· Aggregation: $\mathbf{a}_v^{(l)} \leftarrow \text{AGG}(\{\mathbf{m}_{vu}^{(l)} | u \in \mathcal{N}_v\})$, $\forall v \in V$.

· Update: $\mathbf{h}_v^{(l)} \leftarrow \text{UPT}(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)})$, $\forall v \in V$.

where $\mathcal{N}_v$ is set of neighbors of v.

– 5.3.2. Truyền thông điệp GNN. Truyền thông điệp là khuôn khổ được sử dụng rộng rãi nhất để xây dựng GNN (Gilmer & cộng sự, 2017). Với đồ thị $G = (V, E, X)$, khuôn khổ truyền thông điệp mã hóa mỗi nút $v \in V$ bằng 1 biểu diễn vectơ $\mathbf{h}_v$ & tiếp tục cập nhật biểu diễn nút này bằng cách thu thập lặp lại các biểu diễn của các nút lân cận & áp dụng các lớp mạng nơ-ron để thực hiện phép biến đổi phi tuyến tính của các tập hợp đó:

1. Khởi tạo các biểu diễn vectơ của nút dưới dạng các thuộc tính của nút: $\mathbf{h}_v^{(0)} \leftarrow X_v$, $\forall v \in V$.
2. Cập nhật từng biểu diễn nút dựa trên việc truyền thông điệp qua cấu trúc đồ thị. Ở lớp lth, $l \in [L]$, thực hiện các bước sau:

   · Message: $\mathbf{m}_{vu}^{(1)} \leftarrow \text{MSG}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)})$, $\forall (u,v) \in E$

   · Aggregation: $\mathbf{a}_v^{(l)} \leftarrow \text{AGG}(\{\mathbf{m}_{vu}^{(l)} | u \in \mathcal{N}_v\})$, $\forall v \in V$.

   · Cập nhật: $\mathbf{h}_v^{(l)} \leftarrow \text{UPT}(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^{(l)})$, $\forall v \in V$.

   trong đó $\mathcal{N}_v$ là tập hợp các lân cận của v.

Operations MSG, AGG, & UPT can be implemented via neural networks. Typically, MSG is implemented by a feedforward NN, e.g., $\text{MSG}(p, q) = \sigma(pW_1 + qW_2)$, where W_1, W_2 are learnable weights, & $\sigma(\cdot)$ is an elementwise nonlinear activation. UPT can be chosen in a similar way as MSG. AGG differs as its input is a multiset of vectors & thus order of these vectors should not affect output. AGG is typically implemented as an invariant pooling. Each layer k can have different parameters from other layers. Denote GNNs that follow this message passing framework as MP-GNN.

– Các phép toán MSG, AGG, & UPT có thể được triển khai thông qua mạng nơ-ron. Thông thường, MSG được triển khai bởi 1 mạng nơ-ron truyền thẳng, ví dụ: $\text{MSG}(p, q) = \sigma(pW_1 + qW_2)$, trong đó W_1, W_2 là các trọng số có thể học được, & $\sigma(\cdot)$ là 1 phép kích hoạt phi tuyến tính từng phần tử. UPT có thể được chọn tương tự như MSG. AGG khác ở chỗ đầu vào của nó là 1 tập hợp đa vectơ & do đó thứ tự của các vectơ này không ảnh hưởng đến đầu ra. AGG thường được triển khai dưới dạng 1 phép gộp bất biến. Mỗi lớp k có thể có các tham số khác nhau so với các lớp khác. Ký hiệu các GNN tuân theo khuôn khổ truyền thông điệp này là MP-GNN.

MP-GNN produces representations of all nodes $\{\mathbf{h}_v^{(L)} | v \in V\}$. Each node representation is essentially determined by a subtree rooted at this node Fig. 5.7: Computing flow of MP-GNN to obtain a node representation. Given a specific graph representation learning problem, e.g., classifying a set of nodes $S \subseteq V$, may use representations of relevant nodes in S to make prediction: (5.4)

$$\hat{y}_S = \text{READOUT}(\{\mathbf{h}_v^{(L)} | v \in S\}),$$

where READOUT operation is often implemented via another invariant pooling when $|S| > 1$ plus a feedforward NN to predict target. Combining (11.45) $\mathbf{m}_{ij} = f_{\text{msg}}(\mathbf{h}_i - \mathbf{h}_j)$, $\forall (i,j) \in \tilde{\mathcal{E}}^t$ & (5.4), MP-GNN builds a GNN model for graph representation learning: (5.5)

$$\hat{y}_S = f_{\text{MPGNN}}(G, S).$$

– MP-GNN tạo ra biểu diễn của tất cả các nút $\{\mathbf{h}_v^{(L)} | v \in V\}$. Mỗi biểu diễn nút về cơ bản được xác định bởi 1 cây con có gốc tại nút này Hình 5.7: Luồng tính toán của MP-GNN để thu được biểu diễn nút. Với 1 bài toán học biểu diễn đồ thị cụ thể, ví dụ, phân loại 1 tập hợp các nút $S \subseteq V$, có thể sử dụng biểu diễn của các nút liên quan trong S để đưa ra dự đoán: (5.4)

$$\hat{y}_S = \text{READOUT}(\{\mathbf{h}_v^{(L)} | v \in S\}),$$

trong đó thao tác READOUT thường được triển khai thông qua 1 nhóm bất biến khác khi $|S| > 1$ cộng với 1 mạng nơ-ron truyền thẳng để dự đoán mục tiêu. Kết hợp (11.45) $\mathbf{m}_{ij} = f_{\text{msg}}(\mathbf{h}_i - \mathbf{h}_j)$, $\forall (i,j) \in \tilde{\mathcal{E}}^t$ & (5.4), MP-GNN xây dựng 1 mô hình GNN để học biểu diễn đồ thị: (5.5)

$$\hat{y}_S = f_{\text{MPGNN}}(G, S).$$

Can show permutation invariance of MP-GNN by induction over iteration index l.

– Có thể thể hiện tính bất biến hoán vị của MP-GNN bằng phép quy nạp trên chỉ số lặp l.

Theorem 1 (Invariance of MP-GNN)**.** *$f_{\text{MPGNN}}(\cdot, \cdot)$ satisfies permutation invariant as long as AGG & READOUT operations are invariant pooling operations.*

– $f_{\text{MPGNN}}(\cdot, \cdot)$ thỏa mãn bất biến hoán vị miễn là các hoạt động AGG & READOUT là các hoạt động gộp bất biến.

This can be proved trivially by induction.

MP-GNN by default leverages inductive bias that nodes in graph directly affect each other only via their connected edges. Mutual effect between nodes that are not connected by an edge can be captured via paths that connect these nodes via message passing. Indeed, such inductive bias may not match assumptions in a specific application, & MP-GNN may find it hard to capture mutual effect between 2 far-away nodes. However, message-passing framework has several benefits for model implementation & practical deployment. 1st, it directly works on original graph structure & no pre-processing is needed. 2nd, graphs in practice are typically sparse $|E| \ll |V|^2$ & thus MP-GNN is able to scale to very large but sparse graphs. 3rd, each of 3 operations MSG, AGG, & UPT can be computed in parallel across all nodes & edges, which is beneficial for parallel computing platforms e.g. GPUs & map-reduce systems.

– MP-GNN theo mặc định tận dụng độ lệch quy nạp mà các nút trong đồ thị chỉ ảnh hưởng trực tiếp đến nhau thông qua các cạnh được kết nối của chúng. Hiệu ứng tương hỗ giữa các nút không được kết nối bởi 1 cạnh có thể được nắm bắt thông qua các đường dẫn kết nối các nút này thông qua việc truyền thông điệp. Thật vậy, độ lệch quy nạp như vậy có thể không khớp với các giả định trong 1 ứng dụng cụ thể, & MP-GNN có thể gặp khó khăn trong việc nắm bắt hiệu ứng tương hỗ giữa 2 nút ở xa. Tuy nhiên, khuôn khổ truyền thông điệp có 1 số lợi ích cho việc triển khai mô hình & triển khai thực tế. Thứ nhất, nó hoạt động trực tiếp trên cấu trúc đồ thị gốc & không cần xử lý trước. Thứ hai, đồ thị trong thực tế thường thưa thớt $|E| \ll |V|^2$ & do đó MP-GNN có thể mở rộng thành đồ thị rất lớn nhưng thưa thớt. Thứ ba, mỗi trong 3 phép toán MSG, AGG, & UPT có thể được tính toán song song trên tất cả các nút & cạnh, điều này có lợi cho các nền tảng điện toán song song, ví dụ như GPU & hệ thống map-reduce.
Because it is natural & easy to be implemented in practice, most GNN architectures essentially follow MP-GNN framework by adopting specific MSG, AGG, & UPT operations. Representative approaches include InteractionNet (Battaglia et al, 2016), structure2vec (Dai et al, 2016), GCN (Kipf & Welling, 2017a), GraphSAGE (Hamilton et al, 2017b), GAT (Veličković et al, 2018), GIN (Xu et al, 2019d), & many others (Kearnes et al, 2016; Zhang et al, 2018g).
– Vì tính tự nhiên & dễ dàng triển khai trong thực tế, hầu hết các kiến trúc GNN về cơ bản đều tuân theo khuôn khổ MP-GNN bằng cách áp dụng các phép toán MSG, AGG, & UPT cụ thể. Các phương pháp tiêu biểu bao gồm InteractionNet (Battaglia & cộng sự, 2016), structure2vec (Dai & cộng sự, 2016), GCN (Kipf & Welling, 2017a), GraphSAGE (Hamilton & cộng sự, 2017b), GAT (Veličković & cộng sự, 2018), GIN (Xu & cộng sự, 2019d), & nhiều phương pháp khác (Kearnes & cộng sự, 2016; Zhang & cộng sự, 2018g).

∗ 5.3.3. Expressive Power of MP-GNN. In this section, introduce expressive power of MP-GNN, following results proposed in Xu et al (2019d); Morris et al (2019).
p. 73+++

- 6. GNNs: Scalability.
- 7. Interpretability in GNNs.
- 8. GNNs: Adversarial Robustness.

PART III. FRONTIERS OF GRAPH NEURAL NETWORKS.

- 9. Graph Neural Networks: Graph Classification.
- 10. Graph Neural Networks: Link Prediction.
- 11. GNNs: Graph Generation.
- 12. GNNs: Graph Transformation.
- 13. GNNs: Graph Matching.
- 14. GNNs: Graph Structure Learning.
- 15. Dynamic GNNs.
- 16. Heterogeneous GNNs.
- 17. GNNs: AutoML.
- 18. GNNs: Self-supervised Learning.

PART IV. BROAD & EMERGING APPLICATIONS WITH GRAPH NEURAL NETWORKS.

- 19. GNNs in Modern Recommender Systems.
- 20. GNNs in Computer Vision.
- 21. GNNs in Natural Language Processing.
- 22. GNNs in Program Analysis.
- 23. GNNs in Software Mining.
- 24. GNN-based Biomedical Knowledge Graph Mining in Drug Development.
- 25. GNNs in Predicting Protein Function & Interactions.
- 26. GNNs in Anomaly Detection.
- 27. GNNs in Urban Intelligence.

1.4 DataCamp/a comprehensive introduction to GNNs. Jul 21, 2022

Learn everything about Graph Neural Networks, including what GNNs are, the different types of graph neural networks, & what they're used for. Plus, learn how to build a Graph Neural Network with Pytorch.

1.4.1 What is a graph

A graph is the type of data structure that contains nodes & edges. A node can be a person, place, or thing, & edges define relationship between nodes. Edges can be directed & undirected based on directional dependencies. Learn about the complex Graph dataset: Jazz Musicians Network. It contains 198 nodes & 2742 edges. In community graph plot below, different colors of nodes represent various communities of Jazz musicians & edges connecting them. There is a web of collaboration where a single musician has relationships within & outside community. Fig. Community Graph Plot by Jazz Musicians Network.

– Đồ thị là loại cấu trúc dữ liệu chứa các nút & cạnh. 1 nút có thể là 1 người, địa điểm hoặc vật, & cạnh xác định mối quan hệ giữa các nút. Các cạnh có thể có hướng & vô hướng dựa trên các phụ thuộc hướng. Tìm hiểu về tập dữ liệu Đồ thị phức tạp: Jazz Musicians Network. Tập dữ liệu này chứa 198 nút & 2742 cạnh. Trong biểu đồ cộng đồng bên dưới, các màu khác nhau của các nút đại diện cho các cộng đồng nhạc sĩ Jazz khác nhau & cạnh kết nối họ. Có 1 mạng lưới cộng tác, trong đó 1 nhạc sĩ có các mối quan hệ bên trong & bên ngoài cộng đồng. Hình. Biểu đồ Cộng đồng của Jazz Musicians Network.

Graphs are excellent in dealing with complex problems with relationships & interactions. They are used in pattern recognition, social networks analysis, recommendation systems, & semantic analysis. Creating graph-based solutions is a whole new field that offers rich insights into complex & interlinked datasets.

– Biểu đồ rất hữu ích trong việc giải quyết các vấn đề phức tạp liên quan đến mối quan hệ & tương tác. Chúng được sử dụng trong nhận dạng mẫu, phân tích mạng xã hội, hệ thống đề xuất & phân tích ngữ nghĩa. Việc tạo ra các giải pháp dựa trên biểu đồ là 1 lĩnh vực hoàn toàn mới, cung cấp những hiểu biết sâu sắc về các tập dữ liệu phức tạp & liên kết với nhau.

1.4.2 Graphs with NetworkX

Learn to create a graph using NetworkX. Code below is influenced by Daniel Holmberg's blog on Graph Neural Networks in Python.

1. Create networkx's `DiGraph` object `H`.
2. Add nodes that contain different labels, colors, & size.
3. Add edges to create a relationship between 2 nodes. E.g., $(0, 1)$ means 0 has a directional dependency on 1. Will create bidirectional relationships by adding $(1, 0)$.
4. Extract colors & sizes in form of lists.
5. Plot graph using networkx's `draw` function.

In next step, convert data structure from directional to an undirected graph using `to_undirected()` function.

```
#converting to undirected graph
G = H.to_undirected()
nx.draw(G, with_labels=True, node_color=colors, node_size=sizes)
```

– Học cách tạo đồ thị bằng NetworkX. Mã bên dưới được lấy cảm hứng từ blog của Daniel Holmberg về Mạng Nơ-ron Đồ thị trong Python.

1. Tạo đối tượng `DiGraph` `H` của networkx.
2. Thêm các nút chứa các nhãn, màu sắc, & kích thước khác nhau.
3. Thêm các cạnh để tạo mối quan hệ giữa 2 nút. Ví dụ: $(0, 1)$ nghĩa là 0 có phụ thuộc hướng vào 1. Sẽ tạo mối quan hệ hai chiều bằng cách thêm $(1, 0)$.
4. Trích xuất màu sắc & kích thước dưới dạng danh sách.
5. Vẽ đồ thị bằng hàm `draw` của networkx.

Trong bước tiếp theo, chuyển đổi cấu trúc dữ liệu từ đồ thị có hướng sang đồ thị vô hướng bằng hàm `to_undirected()`.

```
#chuyển đổi sang đồ thị vô hướng
G = H.to_undirected()
nx.draw(G, with_labels=True, node_color=colors, node_size=sizes)
```

1.4.3 Why is it hard to analyze a graph?

Graph-based data structures have drawbacks, & data scientists must understand them before developing graph-based solutions.

1. A graph exists in non-euclidean space. It does not exist in 2D or 3D space, which makes it harder to interpret data. To visualize structure in 2D space, must use various dimensionality reduction tools.
2. Graphs are dynamic; they do not have a fixed form. There can be 2 visually different graphs, but they might have similar adjacency matrix representations. It makes it difficult for us to analyze data using traditional statistical tools.

3. Large size & dimensionality will increase graph's complexity for human interpretations. Dense structure with multiple nodes & thousands of edges is harder to understand & extract insights.

– Cấu trúc dữ liệu dạng đồ thị có những nhược điểm, & các nhà khoa học dữ liệu phải hiểu rõ chúng trước khi phát triển các giải pháp dựa trên đồ thị.

1. Đồ thị tồn tại trong không gian phi Euclid. Nó không tồn tại trong không gian 2D hoặc 3D, điều này khiến việc diễn giải dữ liệu trở nên khó khăn hơn. Để trực quan hóa cấu trúc trong không gian 2D, cần sử dụng nhiều công cụ giảm chiều khác nhau.

2. Đồ thị là động; chúng không có dạng cố định. Có thể có 2 đồ thị khác nhau về mặt trực quan, nhưng chúng có thể có biểu diễn ma trận kề tương tự nhau. Điều này khiến chúng ta khó phân tích dữ liệu bằng các công cụ thống kê truyền thống.

3. Kích thước & chiều lớn sẽ làm tăng độ phức tạp của đồ thị đối với việc diễn giải của con người. Cấu trúc dày đặc với nhiều nút & hàng nghìn cạnh sẽ khó hiểu & rút ra thông tin chi tiết.

1.4.4 What is a GNN?

GNNs are special types of neural networks capable of working with a graph data structure. They are highly influenced by Convolutional Neural Networks (CNNs) & graph embedding. GNNs are used in predicting nodes, edges, & graph-based tasks.

- CNNs are used for image classification. Similarly, GNNs are applied for graph structure (grid of pixels) to predict a class.

- RNNs are used in text classification. Similarly, GNNs are applied to graph structures where every word is a node in a sentence.

GNNs were introduced when Convolutional Neural Networks failed to achieve optimal results due to arbitrary size of graph & complex structure.

– GNN là 1 loại mạng nơ-ron đặc biệt có khả năng làm việc với cấu trúc dữ liệu đồ thị. Chúng chịu ảnh hưởng rất lớn từ Mạng Nơ-ron Tích chập (CNN) & nhúng đồ thị. GNN được sử dụng để dự đoán các nút, cạnh & các tác vụ dựa trên đồ thị.

- CNN được sử dụng để phân loại hình ảnh. Tương tự, GNN được áp dụng cho cấu trúc đồ thị (lưới pixel) để dự đoán 1 lớp.

- RNN được sử dụng trong phân loại văn bản. Tương tự, GNN được áp dụng cho các cấu trúc đồ thị mà mỗi từ là 1 nút trong câu.

GNN được giới thiệu khi Mạng Nơ-ron Tích chập không đạt được kết quả tối ưu do kích thước đồ thị tùy ý & cấu trúc phức tạp.

Input graph is passed through a series of neural networks. Input graph structure is converted into graph embedding, allowing us to maintain information on nodes, edges, & global context. Then feature vector of nodes A, C is passed through neural network layer. It aggregates these features & passes them to next layer. https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-

– Đồ thị đầu vào được truyền qua 1 chuỗi mạng nơ-ron. Cấu trúc đồ thị đầu vào được chuyển đổi thành dạng nhúng đồ thị, cho phép chúng ta duy trì thông tin về các nút, cạnh & ngữ cảnh toàn cục. Sau đó, vectơ đặc trưng của các nút A & C được truyền qua lớp mạng nơ-ron. Nó tổng hợp các đặc trưng này & truyền chúng sang lớp tiếp theo. https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications.

Types of GNNs. There are several types of neural networks, & most of them have some variation of CNNs. In this sect, will be learning about most popular GNNs.

- **Graph Convolutional Networks (GCNs)** are similar to traditional CNNs. It learns features by inspecting neighboring nodes. GNNs aggregate node vectors, pass result to dense layer, & apply nonlinearity using activation function. In short, it consists of Graph convolution, linear layer, & non-learner activation function. There are 2 major types of GCNs: Spatial Convolutional Networks & Spectral Convolutional Networks.

- **Graph Auto-Encoder Networks** learn graph representation using an encoder & attempt to reconstruct input graphs using a decoder. Encoder & decoders are joined by a bottle layer. They are commonly used in link prediction as Auto-Encoders are good at dealing with class balance.

- **Recurrent Graph Neural Networks (RGNNs)** learn best diffusion pattern, & they can handle multi-relational graphs where a single node has multiple relations. This type of GNN uses regularizers to boost smoothness & eliminate over-parameterization. RGNNs use less computation power to produce better results. They are used in generating text, machine translation, speech recognition, generating image descriptions, video tagging, & text summarization.

- **Gated Graph Neural Networks (GGNNs)** are better than RGNNs in performing tasks with long-term dependencies. Gated Graph Neural Networks improve Recurrent Graph Neural Networks by adding a node, edge, & time gates on long-term dependencies. Similar to Gated Recurrent Units (GRUs), gates are used to remember & forget information in different states.

– Có 1 số loại mạng nơ-ron, & hầu hết đều có 1 số biến thể của CNN. Trong phần này, chúng ta sẽ tìm hiểu về các GNN phổ biến nhất.

- **Mạng Tích chập Đồ thị (GCN)** tương tự như CNN truyền thống. Nó học các đặc trưng bằng cách kiểm tra các nút lân cận. GNN tổng hợp các vectơ nút, truyền kết quả đến lớp dày đặc, & áp dụng tính phi tuyến tính bằng hàm kích hoạt. Tóm lại, nó bao gồm tích chập Đồ thị, lớp tuyến tính, & hàm kích hoạt không học. Có 2 loại GCN chính: Mạng Tích chập Không gian & Mạng Tích chập Phổ.
- **Mạng Tự động Mã hóa Đồ thị** học biểu diễn đồ thị bằng bộ mã hóa & cố gắng tái tạo đồ thị đầu vào bằng bộ giải mã. Bộ mã hóa & bộ giải mã được nối với nhau bằng 1 lớp chai. Chúng thường được sử dụng trong dự đoán liên kết vì Bộ tự động Mã hóa rất hiệu quả trong việc xử lý cân bằng lớp.
- **Mạng Nơ-ron Đồ thị Hồi quy (RGNN)** học mẫu khuếch tán tốt nhất, & chúng có thể xử lý đồ thị đa quan hệ, trong đó 1 nút có nhiều quan hệ. Loại GNN này sử dụng các bộ điều chỉnh để tăng độ mượt & loại bỏ tham số hóa quá mức. RGNN sử dụng ít năng lực tính toán hơn để tạo ra kết quả tốt hơn. Chúng được sử dụng trong việc tạo văn bản, dịch máy, nhận dạng giọng nói, tạo mô tả hình ảnh, gắn thẻ video, & tóm tắt văn bản.
- **Mạng Nơ-ron Đồ thị Có Cổng (GGNN)** tốt hơn RGNN trong việc thực hiện các tác vụ có phụ thuộc dài hạn. Mạng Nơ-ron Đồ thị Có Cổng cải thiện Mạng Nơ-ron Đồ thị Hồi quy bằng cách thêm 1 nút, cạnh, & cổng thời gian vào các phụ thuộc dài hạn. Tương tự như Đơn vị Hồi quy Có Cổng (GRU), cổng được sử dụng để ghi nhớ & quên thông tin ở các trạng thái khác nhau.

Types of GNNs tasks. Outline some of types of GNN tasks with examples:

- **Graph Classification.** Use this to classify graphs into various categories. Its applications are social network analysis & text classification.
- **Node Classification.** This task uses neighboring node labels to predict missing node labels in a graph.
- **Link Prediction.** predicts link between a pair of nodes in a graph with an incomplete adjacency matrix. It is commonly used for social networks.
- **Community Detection.** divides nodes into various clusters based on edge structure. It learns from edge weights, & distance & graph objects similarly.
- **Graph Embedding** maps graphs into vectors, preserving relevant information on nodes, edges, & structure.
- **Graph Generation.** learns from sample graph distribution to generate a new but similar graph structure.

– Phác thảo 1 số loại tác vụ GNN kèm ví dụ:

- **Phân loại đồ thị.** Sử dụng tác vụ này để phân loại đồ thị thành nhiều loại khác nhau. Ứng dụng của nó là phân tích mạng xã hội & phân loại văn bản.
- **Phân loại nút.** Tác vụ này sử dụng nhãn nút lân cận để dự đoán nhãn nút bị thiếu trong đồ thị.
- **Dự đoán liên kết.** dự đoán liên kết giữa 1 cặp nút trong đồ thị có ma trận kề không đầy đủ. Nó thường được sử dụng cho mạng xã hội.
- **Phát hiện cộng đồng.** chia các nút thành các cụm khác nhau dựa trên cấu trúc cạnh. Nó học từ trọng số cạnh, & khoảng cách & các đối tượng đồ thị 1 cách tương tự.
- **Nhúng đồ thị** ánh xạ đồ thị thành các vectơ, bảo toàn thông tin liên quan về các nút, cạnh, & cấu trúc.
- **Tạo đồ thị.** học từ phân phối đồ thị mẫu để tạo ra 1 cấu trúc đồ thị mới nhưng tương tự.

Disadvantages of GNNs. There are a few drawbacks to using GNNs. Understanding them will help us determine when to use GNNs & how to optimize performance of our ML models.

1. Most neural networks can go deep to obtain better performance, whereas *GNNs are shallow networks* mostly with 3 layers. It limits us from achieving state-of-art performance on large datasets.
2. *Graph structures are constantly changing*, making it harder to train a model on it.
3. Deploying model to production faces *scalability issues* as these networks are computationally expensive. If have a large & complex graph structure, it will be hard for you to scale GNNs in production.

– Có 1 vài nhược điểm khi sử dụng GNN. Hiểu rõ những nhược điểm này sẽ giúp chúng ta xác định thời điểm sử dụng GNN & cách tối ưu hóa hiệu suất của các mô hình ML.

1. Hầu hết các mạng nơ-ron đều có thể đi sâu để đạt được hiệu suất tốt hơn, trong khi *GNN là mạng nông* chủ yếu có 3 lớp. Điều này hạn chế chúng ta đạt được hiệu suất tiên tiến trên các tập dữ liệu lớn.
2. *Cấu trúc đồ thị liên tục thay đổi*, khiến việc huấn luyện mô hình trên đó trở nên khó khăn hơn.
3. Việc triển khai mô hình lên môi trường sản xuất gặp phải *vấn đề về khả năng mở rộng* vì các mạng này tốn kém về mặt tính toán. Nếu có cấu trúc đồ thị lớn & phức tạp, bạn sẽ khó có thể mở rộng GNN trong môi trường sản xuất.

1.4.5 What is a Graph Convolutional Network (GCN)?

Majority of GNNs are Graph Convolutional Networks, & important to learn about them before jumping into a node classification tutorial. *Convolution* in GCN is same as a convolution in CNNs. It multiplies neurons with weights (filters) to learn from data features. It acts as sliding windows on whole images to learn features from neighboring cells. Filter uses weight sharing to learn various facial features in image recognition systems.

– Phần lớn GNN là Mạng Tích chập Đồ thị, & điều quan trọng là phải tìm hiểu về chúng trước khi bắt đầu hướng dẫn phân loại nút. Tích chập trong GCN tương tự như tích chập trong CNN. Nó nhân các nơ-ron với trọng số (bộ lọc) để học từ các đặc điểm dữ liệu. Nó hoạt động như các cửa sổ trượt trên toàn bộ hình ảnh để học các đặc điểm từ các ô lân cận. Bộ lọc sử dụng chia sẻ trọng số để học các đặc điểm khuôn mặt khác nhau trong các hệ thống nhận dạng hình ảnh.

Now transfer same functionality to GCNs where a model learns features from neighboring nodes. Major difference between GCN & GNN: it is developed to work on non-euclidean data structures where order of nodes & edges can vary.

– Bây giờ hãy chuyển chức năng tương tự sang GCN, nơi mô hình học các đặc điểm từ các nút lân cận. Sự khác biệt chính giữa GCN & GNN: nó được phát triển để hoạt động trên các cấu trúc dữ liệu phi Euclid, trong đó thứ tự các nút & cạnh có thể thay đổi.

There are 2 types of GCNs:

- **Spatial Graph Convolutional Networks** use spatial features to learn from graphs that are located in spatial space.
- **Spectral Graph Convolutional Networks** use Eigen-decomposition of graph Laplacian matrix for information propagation along nodes. These networks were inspired by wave propagation in signals & systems.

– Có 2 loại GCN:

- **Mạng tích chập đồ thị không gian** sử dụng các đặc trưng không gian để học từ các đồ thị nằm trong không gian không gian.
- **Mạng tích chập đồ thị phổ** sử dụng phép phân tích riêng của ma trận Laplacian đồ thị để truyền thông tin dọc theo các nút. Các mạng này được lấy cảm hứng từ sự truyền sóng trong tín hiệu & hệ thống.

1.4.6 How do GNNs work? Build a GNN with PyTorch

Build & train Spectral Graph Convolution for a node classification model. Source code is available at DataLab workbook to experience & run 1st graph-based ML model. Coding examples are influenced by PyTorch Geometric documentation. Install PyTorch https://www.datacamp.com/courses/introduction-to-deep-learning-with-pytorch package as `pytorch_geometric` is built upon it.

```
!pip install -q torch
```

Then use torch version to install `torch-scatter, torch-sparse`. After that, install `pytorch_geometric`'s https://pytorch-geometri readthedocs.io/en/latest/notes/colabs.html latest release from GitHub.

```
%%capture
import os
import torch
os.environ['TORCH'] = torch.__version__
os.environ['PYTHONWARNINGS'] = "ignore"
!pip install torch-scatter -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install torch-sparse -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install git+https://github.com/pyg-team/pytorch_geometric.git
```

Planetoid Cora Dataset. Planetoid is a citation network dataset from Cora, CiteSeer, & PubMed. Nodes are documented with 1433-dimensional bag-of-words feature vectors, & edges are citation links between research papers. There are 7 classes, & will train model to predict missing labels. Will ingest Planetoid Cora dataset, & *row normalize* bag of words input feature. After that, analyze dataset & 1st graph object.

– Planetoid là 1 tập dữ liệu mạng trích dẫn từ Cora, CiteSeer, & PubMed. Các nút được ghi lại bằng các vectơ đặc trưng túi từ 1433 chiều, & các cạnh là các liên kết trích dẫn giữa các bài báo nghiên cứu. Có 7 lớp, & sẽ huấn luyện mô hình để dự đoán các nhãn bị thiếu. Sẽ nhập tập dữ liệu Planetoid Cora, & tính năng đầu vào túi từ *row normalize*. Sau đó, phân tích tập dữ liệu & đối tượng đồ thị đầu tiên.

```
from torch_geometric.datasets import Planetoid
from torch_geometric.transforms import NormalizeFeatures

dataset = Planetoid(root='data/Planetoid', name='Cora', transform=NormalizeFeatures())

print(f'Dataset: {dataset}:')
print('======================')
print(f'Number of graphs: {len(dataset)}')
```

```
print(f'Number of features: {dataset.num_features}')
print(f'Number of classes: {dataset.num_classes}')

data = dataset[0]  # Get the first graph object.
print(data)
```

Cora dataset has 2708 nodes, 10,556 edges, 1433 features, & 7 classes. 1st object has 2708 train, validation, & test masks. Will use these masks to train & evaluate model.

– Tập dữ liệu Cora có 2708 nút, 10.556 cạnh, 1433 đặc trưng, & 7 lớp. Đối tượng đầu tiên có 2708 mặt nạ huấn luyện, xác thực, & kiểm tra. Sẽ sử dụng các mặt nạ này để huấn luyện & đánh giá mô hình.

```
Dataset: Cora():
======================
Number of graphs: 1
Number of features: 1433
Number of classes: 7
Data(x=[2708, 1433], edge_index=[2, 10556], y=[2708], train_mask=[2708], val_mask=[2708], test_mask=[2708])
```

Node classification with GNN. Create a GCN model structure that contains 2 GCNConv https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.GCNConv layers `relu` & a dropout rate of 0.5. Model consists of 16 hidden channels.

– Tạo cấu trúc mô hình GCN chứa 2 lớp GCNConv https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.GCNConv `relu` & tỷ lệ bỏ qua là 0,5. Mô hình bao gồm 16 kênh ẩn.

GCN layer:

$$\mathbf{x}_v^{(l+1)} = \mathbf{W}^{(l+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(l)}.$$

$W^{(l+1)}$ is a trainable weight matrix in above equation & $C_{w,v}$ denotes to a fixed normalization coefficient for each edge.

– $W^{(l+1)}$ là ma trận trọng số có thể đào tạo được trong phương trình trên & $C_{w,v}$ biểu thị cho hệ số chuẩn hóa cố định cho mỗi cạnh.

1.5 Geeks4Geeks/what are GNNs?

1.6 Geeks4Geeks/GNNs with PyTorch

https://www.geeksforgeeks.org/deep-learning/graph-neural-networks-with-pytorch/.

GNNs represent a powerful class of ML models tailored for interpreting data described by graphs. This is particularly useful because many real-world structures are networks composed on interconnected elements, e.g. social networks, molecular structures, & communication systems. See how can use PyTorch for building GNNs.

1.6.1 Implementation of simple GNN model using PyTorch

Implementing Graph Neural Networks (GNNs) with CORA dataset in PyTorch, specifically using PyTorch Geometric (PyG), involves several steps. A guide through process, including code snippets for each step.

– Việc triển khai Mạng Nơ-ron Đồ thị (GNN) với tập dữ liệu CORA trong PyTorch, cụ thể là sử dụng PyTorch Geometric (PyG), bao gồm nhiều bước. Hướng dẫn chi tiết quy trình, bao gồm các đoạn mã cho từng bước.

1. **Loading CORA Dataset.** CORA dataset is a citation graph where nodes represent documents, & edges represent citation links. Each document is classified into 1 of 7 categories, making it a popular dataset for node classification tasks in GNNs. 1st, ensure have PyTorch Geometric installed. if not, install it using `pip`:

   – **Đang tải Bộ dữ liệu CORA.** Bộ dữ liệu CORA là 1 biểu đồ trích dẫn, trong đó các nút đại diện cho tài liệu, & cạnh đại diện cho liên kết trích dẫn. Mỗi tài liệu được phân loại thành 1 trong 7 loại, khiến nó trở thành 1 bộ dữ liệu phổ biến cho các tác vụ phân loại nút trong GNN. Trước tiên, hãy đảm bảo đã cài đặt PyTorch Geometric. Nếu chưa, hãy cài đặt bằng `pip`:

   ```
   pip install torch-scatter torch-sparse torch-cluster torch-spline-conv torch-geometric
   ```

   Then load CORA dataset:

   ```
   import torch
   from torch_geometric.data import Planetoid
   import torch.nn.functional as F
   from torch_geometric.nn import GCNConv

   # load Cora dataset
   dataset = Planetoid(root = 'data/Planetoid', name = 'Cora')
   ```

2. **Defining a GNN Model Using PyTorch.** Define a simple GNN model using 1 of most straightforward types of GNN layers, Graph Convolutional Network (GCN) layer, provided by PyTorch Geometric.

   – **Định nghĩa mô hình GNN bằng PyTorch.** Định nghĩa mô hình GNN đơn giản bằng 1 trong những loại lớp GNN đơn giản nhất, lớp Mạng tích chập đồ thị (GCN), do PyTorch Geometric cung cấp.

   A custom Graph Neural Network (GNN) model is built using PyTorch's `torch.nn.Module` class. Model consists of 2 Graph Convolutional Network (GCN) layers, each followed by a Rectified Linear Unit (ReLU) activation function & dropout regularization. Model's 'forward' method takes feature data & edge information as input, applies defined layers sequentially, & outputs a log-softmax activation for classification. Additionally, an Adam optimizer is initialized to train model with a specified learning rate & weight decay.

   – 1 mô hình Mạng Nơ-ron Đồ thị (GNN) tùy chỉnh được xây dựng bằng lớp `torch.nn.Module` của PyTorch. Mô hình bao gồm 2 lớp Mạng Tích chập Đồ thị (GCN), mỗi lớp được theo sau bởi 1 hàm kích hoạt Đơn vị Tuyến tính Chỉnh lưu (ReLU) & chính quy hóa dropout. Phương thức 'forward' của mô hình lấy dữ liệu đặc trưng & thông tin cạnh làm đầu vào, áp dụng các lớp đã xác định theo trình tự, & xuất ra 1 phép kích hoạt log-softmax để phân loại. Ngoài ra, 1 trình tối ưu hóa Adam được khởi tạo để huấn luyện mô hình với tốc độ học & suy giảm trọng số được chỉ định.

3. **Training GNN model on CORA dataset.** Train model. This involves initializing model, defining optimizer, & running training loop.

   – **Huấn luyện mô hình GNN trên tập dữ liệu CORA.** Huấn luyện mô hình. Quá trình này bao gồm khởi tạo mô hình, xác định trình tối ưu hóa & chạy vòng lặp huấn luyện.

4. **Evaluating Model's Performance.** This code defines a basic GNN model, trains it on CORA dataset, & evaluates its accuracy. Model architecture & training parameters are kept simple for demonstration purposes. In practice, may want to experiment with deeper models, different types of GNN layers, & other optimization techniques to improve performance.

   – **Đánh giá Hiệu suất Mô hình.** Mã này định nghĩa 1 mô hình GNN cơ bản, huấn luyện nó trên tập dữ liệu CORA, & đánh giá độ chính xác của nó. Kiến trúc mô hình & các tham số huấn luyện được giữ đơn giản cho mục đích trình diễn. Trong thực tế, có thể bạn sẽ muốn thử nghiệm với các mô hình sâu hơn, các loại lớp GNN khác nhau, & các kỹ thuật tối ưu hóa khác để cải thiện hiệu suất.

Output shows loss value decreasing over 200 epochs of training a GNN on CORA dataset, indicating: model is learning to classify nodes more accurately over time. Final test accuracy of 0.811 demonstrates: trained GNN model can correctly predict class of over 81% of nodes in test set, showcasing its effectiveness in node classification tasks within graph-structured data.

– Kết quả đầu ra cho thấy giá trị mất mát giảm dần qua 200 kỷ nguyên huấn luyện GNN trên tập dữ liệu CORA, cho thấy: mô hình đang học cách phân loại các nút chính xác hơn theo thời gian. Độ chính xác kiểm tra cuối cùng là 0,811 cho thấy: mô hình GNN đã huấn luyện có thể dự đoán chính xác lớp của hơn 81% số nút trong tập kiểm tra, thể hiện hiệu quả của nó trong các tác vụ phân loại nút trong dữ liệu có cấu trúc đồ thị.

1.7 Geeks4Geeks/What is PyTorch?

Resources – Tài nguyên.

- Geeks for Geeks/what is PyTorch?

PyTorch is a DL library built on Python. It provides GPU acceleration, dynamic computation graphs & an intuitive interface for DL researchers & developers. PyTorch follows a "define-by-run" approach, i.e., its computational graphs are constructed on fly allowing for better debugging & model customization.

– PyTorch là 1 thư viện DL được xây dựng trên Python. Nó cung cấp khả năng tăng tốc GPU, đồ thị tính toán động & giao diện trực quan cho các nhà nghiên cứu DL & nhà phát triển. PyTorch áp dụng phương pháp "xác định bằng chạy", i.e., các đồ thị tính toán của nó được xây dựng trực tiếp, cho phép gỡ lỗi tốt hơn & tùy chỉnh mô hình.

1.7.1 Key features of PyTorch

- Use dynamic graphs allowing flexibility in model execution & debugging.
- Provide an automatic differentiation engine that simplifies gradient computation for DL.
- Support CUDA allowing computations to be performed efficiently on GPUs.

PyTorch can be installed on Windows, macOS & Linux using pip for CPU (without GPU):

```
!pip install torch torchvision torchaudio
```

1.7.2 PyTorch tensors

Tensors are fundamental data structures in PyTorch, similar to NumPy arrays but with GPU acceleration capabilities. PyTorch tensors support automatic differentiation, making them suitable for DL tasks.

1. Operations on tensors: element-wise addition, matrix multiplication
2. Reshape & transpose tensors
3. Autograd & computational graphs: `autograd` module automates gradient calculation for backpropagation. This is crucial in training deep neural networks.

   PyTorch dynamically creates a computational graph that tracks operations & gradients for backpropagation.

– Tensor là cấu trúc dữ liệu cơ bản trong PyTorch, tương tự như mảng NumPy nhưng có khả năng tăng tốc GPU. Tensor PyTorch hỗ trợ tính vi phân tự động, phù hợp cho các tác vụ DL.

1. Các phép toán trên tensor: phép cộng từng phần tử, phép nhân ma trận
2. Định hình lại & chuyển vị tensor
3. Đồ thị tính toán Autograd &: Mô-đun `autograd` tự động tính toán gradient cho phép lan truyền ngược. Điều này rất quan trọng trong việc huấn luyện mạng nơ-ron sâu.

   PyTorch tạo động 1 đồ thị tính toán theo dõi các phép toán & gradient cho phép lan truyền ngược.

1.7.3 Building neural networks in PyTorch

In PyTorch, neural networks are built using `torch.nn` module where:

- `nn.Linear(in_features, out_features)` defines a fully connected (dense) layer.
- Activation functions like `torch.relu, torch.sigmoid, torch.softmax` are applied between layers.
- `forward()` method defines how data moves through network.

To built a neural network in PyTorch, create a class that inherits from `torch.nn.Module` & defines its layers & forward pass.
– Trong PyTorch, mạng nơ-ron được xây dựng bằng mô-đun `torch.nn`, trong đó:

- `nn.Linear(in_features, out_features)` định nghĩa 1 lớp được kết nối đầy đủ (dense).
- Các hàm kích hoạt như `torch.relu, torch.sigmoid, torch.softmax` được áp dụng giữa các lớp.
- Phương thức
- `forward()` định nghĩa cách dữ liệu di chuyển qua mạng.

Để xây dựng mạng nơ-ron trong PyTorch, hãy tạo 1 lớp kế thừa từ `torch.nn.Module` & định nghĩa các lớp của nó & forward pass.

1.7.4 Define loss function & optimizer

Once define our model, need to specify:

- A loss function to measure error.
- An optimizer to update weights based on computed gradients.

Use `nn.BCELoss()` for binary cross-entropy loss & used `optim.Adam()` for Adam optimizer to combine benefits of momentum & adaptive learning rates.
– Sau khi xác định mô hình, cần chỉ định:

- 1 hàm mất mát để đo lỗi.
- 1 trình tối ưu hóa để cập nhật trọng số dựa trên các gradient đã tính toán.

Sử dụng `nn.BCELoss()` cho mất mát entropy chéo nhị phân & sử dụng `optim.Adam()` cho trình tối ưu hóa Adam để kết hợp lợi ích của động lượng & tốc độ học thích ứng.

1.7.5 Train model

Training involves:

1. Generating dummy data (100 samples, each with 10 features).
2. Running a training loop where
   - `optimizer.zero_grad()` clears accumulated gradients from previous step.
   - Forward Pass `model(inputs)` passes inputs through model to generate predictions.
   - Loss Computation `criterion(outputs, targets)` computes difference between predictions & actual labels.
   - Backpropagation `loss.backward()` computes gradients for all weights.
   - Optimizer Step `optimizer.step()` updates weights based on computed gradients.

– Quá trình huấn luyện bao gồm:

1. Tạo dữ liệu giả (100 mẫu, mỗi mẫu có 10 đặc trưng).
2. Chạy vòng lặp huấn luyện trong đó
   - `optimizer.zero_grad()` xóa các gradient tích lũy từ bước trước.
   - Forward Pass `model(inputs)` truyền các đầu vào qua mô hình để tạo dự đoán.
   - Loss Computation `criteria(outputs, targets)` tính toán sự khác biệt giữa các nhãn dự đoán & nhãn thực tế.
   - Backpropagation `loss.backward()` tính toán gradient cho tất cả các trọng số.
   - Optimizer Step `optimizer.step()` cập nhật các trọng số dựa trên các gradient đã tính toán.

1.7.6 PyTorch vs. TensorFlow

Feature	PyTorch	TensorFlow
Computational graph	dynamic	static (TF 1.x), dynamic (TF 2.0)
ease of use	Pythonic, easy to debug	steeper learning curve
performance	fast with eager execution	optimized for large-scale deployment
Deployment	TorchScript & ONNX	TensorFlow Serving & TensorFlow Lite
Popularity in research	widely used	also widely used but more in production

1.7.7 Applications of PyTorch

1. Computer Vision: PyTorch is widely used in image classification, object detection & segmentation using CNNs & Transformers (e.g., ViT).
2. Natural Language Processing (MLP): PyTorch supports transformers, recurrent neural networks (RNNs) & LSTMs for applications like text generation & sentiment analysis.
3. Reinforcement Learning: PyTorch is used in Deep Q-Networks (DQN), Policy Gradient Methods & Actor-Critic algorithms.

PyTorch is used in industry for computer vision, NLP & reinforcement learning applications. With its strong community support & easy-to-use API, it continues to be 1 of leading DL frameworks.

1. Thị giác máy tính: PyTorch được sử dụng rộng rãi trong phân loại hình ảnh, phát hiện đối tượng & phân đoạn bằng CNN & Bộ biến đổi (ví dụ: ViT).
2. Xử lý ngôn ngữ tự nhiên (MLP): PyTorch hỗ trợ bộ biến đổi, mạng nơ-ron hồi quy (RNN) & LSTM cho các ứng dụng như tạo văn bản & phân tích cảm xúc.
3. Học tăng cường: PyTorch được sử dụng trong Mạng Q sâu (DQN), Phương pháp Gradient chính sách & thuật toán Actor-Critic.

PyTorch được sử dụng trong công nghiệp cho các ứng dụng thị giác máy tính, NLP & học tăng cường. Với sự hỗ trợ cộng đồng mạnh mẽ & API dễ sử dụng, nó tiếp tục là 1 trong những nền tảng DL hàng đầu.

1.8 Geeks for Geeks/Understanding `torch.nn.Parameter`

Resources – Tài nguyên.

- Geeks for Geeks/understanding `torch.nn.Parameter`.

PyTorch is a widely used library for building & training neural networks, & understanding its components is key to effectively using it for ML tasks. 1 of essential classes in PyTorch is `torch.nn.Parameter`, which plays a crucial role in defining trainable parameters within a model. This article will explore what `torch.nn.Parameter` is, its significance, & how it is used in PyTorch models.

– PyTorch là 1 thư viện được sử dụng rộng rãi để xây dựng & huấn luyện mạng nơ-ron, & việc hiểu các thành phần của nó là chìa khóa để sử dụng hiệu quả cho các tác vụ ML. 1 trong những lớp thiết yếu trong PyTorch là `torch.nn.Parameter`, đóng vai trò quan trọng trong việc xác định các tham số có thể huấn luyện trong 1 mô hình. Bài viết này sẽ tìm hiểu `torch.nn.Parameter` là gì, ý nghĩa của nó & cách sử dụng nó trong các mô hình PyTorch.

1.8.1 What is `torch.nn.Parameter`?

`torch.nn.Parameter` is a subclass of `torch.Tensor`, designed specifically for holding parameters in a model that should be considered during training. When a tensor is wrapped with `torch.nn.Parameter`, it automatically becomes a part of model's parameters, & thus it will be updated when backpropagation is applied during training. This is fundamental because it tells PyTorch's optimizer which tensors should be updated through learning processes.

– `torch.nn.Parameter` là 1 lớp con của `torch.Tensor`, được thiết kế đặc biệt để lưu trữ các tham số trong mô hình cần được xem xét trong quá trình huấn luyện. Khi 1 tensor được bao bọc bởi `torch.nn.Parameter`, nó sẽ tự động trở thành 1 phần của các tham số mô hình, & do đó, nó sẽ được cập nhật khi áp dụng lan truyền ngược trong quá trình huấn luyện. Điều này rất quan trọng vì nó cho trình tối ưu hóa của PyTorch biết tensor nào cần được cập nhật thông qua các quy trình học.

1.8.2 Key features of `torch.nn.Parameter`

Here are some key features of `torch.nn.Parameter`:

1. **Trainable Parameters**: By default, parameters wrapped in `torch.nn.Parameter` are considered trainable, i.e., they are part of model's learnable parameters & are subject to updates during gradient descent.
2. **Integration with Modules**: In PyTorch, models are typically built using `torch.nn.Module` class. Any `torch.nn.Module` assigned as an attribute to a module is automatically registered as a parameter of module.
3. **Easy Serialization** : Parameters can be easily saved along with other model components using PyTorch's serialization tools, making it easy to save & load trained models.

– Dưới đây là 1 số tính năng chính của `torch.nn.Parameter`:

1. **Tham số có thể huấn luyện**: Theo mặc định, các tham số được gói trong `torch.nn.Parameter` được coi là có thể huấn luyện, i.e., chúng là 1 phần của các tham số có thể học được của mô hình & có thể được cập nhật trong quá trình giảm dần gradient.
2. **Tích hợp với Mô-đun**: Trong PyTorch, các mô hình thường được xây dựng bằng lớp `torch.nn.Module`. Bất kỳ `torch.nn.Module` nào được gán làm thuộc tính cho 1 mô-đun sẽ tự động được đăng ký làm tham số của mô-đun.
3. **Tuần tự hóa Dễ dàng**: Các tham số có thể dễ dàng được lưu cùng với các thành phần mô hình khác bằng các công cụ tuần tự hóa của PyTorch, giúp việc lưu & tải các mô hình đã huấn luyện trở nên dễ dàng.

1.8.3 Usage of `torch.nn.Parameter`

To understand how `torch.nn.Parameter` is used, consider a simple example where define a custom module with learnable weights & bias.

– Để hiểu cách sử dụng `torch.nn.Parameter`, hãy xem xét 1 ví dụ đơn giản trong đó định nghĩa 1 mô-đun tùy chỉnh với trọng số có thể học được & bias.

```
# understand torch.nn.parameter
import torch.nn as nn

class my_linear(nn.Module):
    def __init__(self, in_features, out_features):
        super(my_linear, self).__init__()
        # define weight & bias parameters
        self.weight = nn.Parameter(torch.randn(out_features, in_features))
        self.bias = nn.Parameter(torch.randn(out_features))

    def forward(self, x): # implement forward pass
        return torch.matmul(x, self.weight.t()) + self.bias
```

In this example, `self.weight, self.bias` are instances of `torch.nn.Parameter`. During training, these parameters will be updated to minimize loss function, thanks to their registration as module parameters.

– Trong ví dụ này, `self.weight, self.bias` là các thể hiện của `torch.nn.Parameter`. Trong quá trình huấn luyện, các tham số này sẽ được cập nhật để tối thiểu hóa hàm mất mát, nhờ vào việc đăng ký chúng dưới dạng tham số mô-đun.

1.8.4 Step-by-step guide to training a model with `torch.nn.Parameter` in PyTorch

This section provides a comprehensive explanation & demonstration of how to use `torch.nn.Parameter` in PyTorch to train a simple neural network. Each step incudes a detailed description along with corresponding code snippets.

1. **Import Necessary Libraries.** 1st import required PyTorch modules for neural network construction & optimization:

```
import torch
import torch.nn as nn
import torch.optim as optim
```

2. **Define Neural Network Class.** Create a custom neural network class `SimpleNet` using `nn.Module`. Define a trainable parameter `self.weight` using `torch.nn.Parameter`.

3. **Instantiate Model.** Create an instance of `SimpleNet`. Print initial model parameters to verify that weight has been initialized.

4. **Set Up Loss Function & Optimizer.** Define loss function & optimizer for training. Here use Mean Squared Error (MSE) loss & Stochastic Gradient Descent (SGD) optimizer.

5. **Prepare Training Data.** Define input & target tensors. These tensors represent data that model will learn from during training.

6. **Train Model.** Execute training loop. This loop involves forwarding pass, loss calculation, backpropagation, & parameter updates. Monitor training process by printing loss & current weight every 10 epochs.

7. **Display Final Model Parameters.** After training, print final learned parameters to see how well model has learned to approximate target from input.

```
print("Final Model Parameters:", list(model.parameters()))
```

– Phần này cung cấp giải thích toàn diện & minh họa cách sử dụng `torch.nn.Parameter` trong PyTorch để huấn luyện 1 mạng nơ-ron đơn giản. Mỗi bước đều bao gồm mô tả chi tiết cùng với các đoạn mã tương ứng.

1. **Nhập các thư viện cần thiết.** Đầu tiên, nhập các mô-đun PyTorch cần thiết để xây dựng mạng nơ-ron & tối ưu hóa:

```
import torch
import torch.nn as nn
import torch.optim as optim
```

2. **Định nghĩa lớp mạng nơ-ron.** Tạo 1 lớp mạng nơ-ron tùy chỉnh `SimpleNet` bằng `nn.Module`. Định nghĩa tham số có thể huấn luyện `self.weight` bằng `torch.nn.Parameter`.

3. **Khởi tạo mô hình.** Tạo 1 phiên bản của `SimpleNet`. In các tham số mô hình ban đầu để xác minh rằng trọng số đã được khởi tạo.

4. **Thiết lập Hàm Mất mát & Bộ Tối ưu hóa.** Định nghĩa hàm mất mát & bộ tối ưu hóa cho huấn luyện. Ở đây, sử dụng bộ tối ưu hóa MSE (Sai Số Bình phương Trung bình) & Bộ Tối ưu hóa SGD (Giảm Gradient Ngẫu nhiên).

5. **Chuẩn bị Dữ liệu Huấn luyện.** Định nghĩa tenxơ đầu vào & mục tiêu. Các tenxơ này biểu diễn dữ liệu mà mô hình sẽ học trong quá trình huấn luyện.

6. **Huấn luyện Mô hình.** Thực hiện vòng lặp huấn luyện. Vòng lặp này bao gồm chuyển tiếp, tính toán mất mát, lan truyền ngược, & cập nhật tham số. Theo dõi quá trình huấn luyện bằng cách in ra giá trị mất mát & trọng số hiện tại sau mỗi 10 kỷ nguyên.

7. **Hiển thị Tham số Mô hình Cuối cùng.** Sau khi huấn luyện, in ra các tham số đã học cuối cùng để xem mô hình đã học được bao nhiêu để ước tính mục tiêu từ đầu vào.

```
print("Final Model Parameters:", list(model.parameters()))
```

1.8.5 Why use `torch.nn.Parameter`?

Using `torch.nn.Parameter` offers several advantages:

- **Explicitness**: It makes it clear which t tensors are intended to be parameters that optimizer should update, improving code readability & maintainability.
- **Convenience**: It simplifies implementation of custom layers & models, as PyTorch handles underlying complexity of parameter updates.
- **Compatibility.** Ensures compatibility with various PyTorch functionalities like optimizers, saving, & loading mechanisms.

– Sử dụng `torch.nn.Parameter` mang lại 1 số lợi thế:

- **Tính rõ ràng**: Nó làm rõ các tenxơ t nào được dự định là tham số mà trình tối ưu hóa nên cập nhật, cải thiện khả năng đọc mã & khả năng bảo trì.
- **Tính tiện lợi**: Nó đơn giản hóa việc triển khai các lớp & mô hình tùy chỉnh, vì PyTorch xử lý được độ phức tạp tiềm ẩn của việc cập nhật tham số.
- **Tính tương thích.** Đảm bảo tính tương thích với nhiều chức năng khác nhau của PyTorch như trình tối ưu hóa, cơ chế lưu, & tải.

1.8.6 Conclusion

Understanding & using `torch.nn.Parameter` is essential for anyone working with PyTorch, especially when implementing custom model components. It provides a structured way to define what parts of a model should learn from data, facilitating development & training of complex neural networks. With `torch.nn.Parameter`, can ensure: your model's parameters are correctly managed & updated through training process, leading to more effective learning outcomes.

– Hiểu & sử dụng `torch.nn.Parameter` là điều cần thiết cho bất kỳ ai làm việc với PyTorch, đặc biệt là khi triển khai các thành phần mô hình tùy chỉnh. Nó cung cấp 1 phương pháp có cấu trúc để xác định những phần nào của mô hình cần học từ dữ liệu, tạo điều kiện thuận lợi cho việc phát triển & huấn luyện các mạng nơ-ron phức tạp. Với `torch.nn.Parameter`, bạn có thể đảm bảo: các tham số của mô hình được quản lý chính xác & cập nhật trong suốt quá trình huấn luyện, mang lại kết quả học tập hiệu quả hơn.

2 GNNs for Combinatorial Optimization

2.1 Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, Petar Veličković. Combinatorial Optimization & Reasoning with GNNs

- Abstract. Combinatorial optimization is a well-established area in operations research & CS. Until recently, its methods have focused on solving problem instances in isolation, ignoring that they often stem from related data distributions in practice. However, recent years have seen a surge of interest in using ML, especially GNNs, as a key building block for combinatorial tasks, either directly as solvers or by enhancing exact solvers. Inductive bias of GNNs effectively encodes combinatorial & relational input due to their invariance to permutations & awareness of input sparsity. This paper presents a conceptual review of recent key advancements in this emerging field, aiming at optimization & ML researchers.

  – Tối ưu hóa tổ hợp là 1 lĩnh vực đã được xác lập rõ ràng trong nghiên cứu vận hành & Khoa học Máy tính. Cho đến gần đây, các phương pháp của nó tập trung vào việc giải quyết các trường hợp bài toán 1 cách riêng lẻ, bỏ qua việc chúng thường xuất phát từ các phân phối dữ liệu liên quan trong thực tế. Tuy nhiên, những năm gần đây đã chứng kiến sự quan tâm ngày càng tăng đối với việc sử dụng máy học (ML), đặc biệt là mạng nơ-ron nhân tạo (GNN), như 1 nền tảng cốt lõi cho các tác vụ tổ hợp, trực tiếp như các bộ giải hoặc bằng cách tăng cường các bộ giải chính xác. Độ lệch quy nạp của GNN mã hóa hiệu quả đầu vào tổ hợp & quan hệ do tính bất biến của chúng đối với các hoán vị & nhận thức được tính thưa thớt của đầu vào. Bài báo này trình bày 1 tổng quan khái niệm về những tiến bộ quan trọng gần đây trong lĩnh vực mới nổi này, hướng đến các nhà nghiên cứu tối ưu hóa & ML.

  Keywords. Combinatorial optimization, graph neural networks, reasoning.

- 1. Introduction. Combinatorial optimization (CO) has developed into an interdisciplinary field spanning optimization, operations research, discrete mathematics, & CS, with many critical real-world applications e.g. vehicle routing or scheduling ; see (Korte & Vygen, 2012) for a general overview. Intuitively, CO deals with problems that involve optimizing a cost (or objective) function by selecting a subset from a finite set, with the latter encoding constraints on solution space. Although CO problems are generally hard from a complexity theory standpoint due to their discrete, non-convex nature (Karp, 1972), many of them are routinely solved in practice. Historically, optimization & theoretical CS communities have been focusing on finding optimal (Korte & Vygen, 2012), heuristic (Boussaïd et al., 2013), or approximate (Vazirani, 2010) solutions for individual problem instances. However, in many practical situations of interest, one often must solve problem instances with specific characteristics or patterns. E.g., a trucking company may solve vehicle routing instances for same city daily, with only slight differences across instances in travel times due to varying traffic conditions. Hence, data-dependent algorithms or ML approaches, which may

exploit these patterns, have recently gained traction in CO field (Bengio et al., 2021; Gasse et al., 2022). Promise: one can develop faster algorithms for practical cases by exploiting common patterns in given instances.

– Tối ưu hóa tổ hợp (CO) đã phát triển thành 1 lĩnh vực liên ngành bao gồm tối ưu hóa, nghiên cứu vận hành, toán học rời rạc, & CS, với nhiều ứng dụng thực tế quan trọng, ví dụ như định tuyến hoặc lập lịch trình xe cộ; xem (Korte & Vygen, 2012) để biết tổng quan chung. Về mặt trực quan, CO xử lý các bài toán liên quan đến việc tối ưu hóa hàm chi phí (hoặc hàm mục tiêu) bằng cách chọn 1 tập con từ 1 tập hữu hạn, với các ràng buộc mã hóa sau trên không gian nghiệm. Mặc dù các bài toán CO thường khó về mặt lý thuyết độ phức tạp do tính chất rời rạc, không lồi của chúng (Karp, 1972), nhưng nhiều bài toán trong số đó đã được giải quyết thường xuyên trong thực tế. Theo truyền thống, các cộng đồng CS lý thuyết & tối ưu hóa đã tập trung vào việc tìm kiếm các giải pháp tối ưu (Korte & Vygen, 2012), giải pháp kinh nghiệm (Boussaïd & cộng sự, 2013) hoặc giải pháp gần đúng (Vazirani, 2010) cho các trường hợp vấn đề riêng lẻ. Tuy nhiên, trong nhiều tình huống thực tế quan tâm, người ta thường phải giải quyết các trường hợp vấn đề với các đặc điểm hoặc mẫu cụ thể. Ví dụ: 1 công ty vận tải có thể giải quyết các trường hợp định tuyến xe cho cùng 1 thành phố hàng ngày, với chỉ 1 số khác biệt nhỏ giữa các trường hợp về thời gian di chuyển do điều kiện giao thông khác nhau. Do đó, các thuật toán phụ thuộc dữ liệu hoặc các phương pháp ML, có thể khai thác các mẫu này, gần đây đã thu hút được sự chú ý trong lĩnh vực CO (Bengio & cộng sự, 2021; Gasse & cộng sự, 2022). Hứa hẹn: người ta có thể phát triển các thuật toán nhanh hơn cho các trường hợp thực tế bằng cách khai thác các mẫu chung trong các trường hợp nhất định.

Due to discrete nature of most CO problems & prevalence of network data in real world, graphs are a central object of study in CO field. E.g., well-known & relevant problems e.g. Traveling Salesperson problem (TSP) & other vehicle routing problems naturally induce a graph structure. In fact, from 21 NP-complete problems identified by Karp (1972), 10 are decision versions of graph optimization problems. Most other ones, e.g. set covering problem, can also be modeled over graphs. Moreover, interaction between variables & constraints in combinatorial optimization problems naturally induces a bipartite graph, i.e., a variable & constraint share an edge if variable appears with a nonzero coefficient in constraint. These graphs commonly exhibit patterns in their structure & features, which ML approaches should exploit.

– Do tính chất rời rạc của hầu hết các vấn đề CO & sự phổ biến của dữ liệu mạng trong thế giới thực, đồ thị là đối tượng nghiên cứu trung tâm trong lĩnh vực CO. E.g., các vấn đề & liên quan nổi tiếng & ví dụ như bài toán Người bán hàng du lịch (TSP) & các vấn đề định tuyến phương tiện khác tự nhiên tạo ra 1 cấu trúc đồ thị. Trên thực tế, trong số 21 bài toán NP-đầy đủ được Karp (1972) xác định, 10 bài toán là phiên bản quyết định của các bài toán tối ưu hóa đồ thị. Hầu hết các bài toán khác, ví dụ như bài toán phủ tập hợp, cũng có thể được mô hình hóa trên đồ thị. Hơn nữa, tương tác giữa các biến & ràng buộc trong các bài toán tối ưu hóa tổ hợp tự nhiên tạo ra 1 đồ thị 2 phía, tức là 1 biến & ràng buộc chia sẻ 1 cạnh nếu biến xuất hiện với hệ số khác không trong ràng buộc. Các đồ thị này thường thể hiện các mẫu trong cấu trúc & các tính năng của chúng, mà các phương pháp ML nên khai thác.

- 1.1. What Are Challenges for ML? There are several critical challenges in successfully applying ML methods within CO, especially for problems involving graphs. Graphs exhibit symmetries, i.e., remaining or reordering nodes does not result in different graphs. Hence, for any ML method dealing with graphs, taking into account invariance to permutation is crucial. Combinatorial optimization problem instances are large & usually sparse, especially those arising from real world. Hence, employed ML method must be scalable & sparsity aware. Simultaneously, employed method has to be expressive enough to detect & exploit relevant patterns in given instance or data distribution. ML method should be capable of handling auxiliary information, e.g. objective & user-defined constraints. Most of current ML approaches are within supervised regime. They require a large amount of training data to optimize model's parameters. In context of CO, this means solving many possibly hard problem instances, which might prohibit application of these approaches in real-world scenarios. Further, ML method has to generalize beyond its training data, e.g., transferring to instances of different sizes.

  – Thách thức đối với ML là gì? Có 1 số thách thức quan trọng trong việc áp dụng thành công các phương pháp ML trong CO, đặc biệt là đối với các bài toán liên quan đến đồ thị. Đồ thị thể hiện tính đối xứng, nghĩa là việc giữ lại hoặc sắp xếp lại các nút không dẫn đến các đồ thị khác nhau. Do đó, đối với bất kỳ phương pháp ML nào xử lý đồ thị, việc tính đến tính bất biến của hoán vị là rất quan trọng. Các trường hợp bài toán tối ưu hóa tổ hợp thường lớn & thưa thớt, đặc biệt là những trường hợp phát sinh từ thế giới thực. Do đó, phương pháp ML được sử dụng phải có khả năng mở rộng & nhận thức được tính thưa thớt. Đồng thời, phương pháp được sử dụng phải đủ biểu đạt để phát hiện & khai thác các mẫu liên quan trong các trường hợp hoặc phân phối dữ liệu nhất định. Phương pháp ML phải có khả năng xử lý thông tin bổ trợ, ví dụ: các ràng buộc khách quan & do người dùng xác định. Hầu hết các phương pháp ML hiện tại đều nằm trong chế độ giám sát. Chúng yêu cầu 1 lượng lớn dữ liệu đào tạo để tối ưu hóa các tham số của mô hình. Trong bối cảnh CO, điều này có nghĩa là phải giải quyết nhiều trường hợp bài toán có thể khó, điều này có thể ngăn cản việc áp dụng các phương pháp này trong các tình huống thực tế. Hơn nữa, phương pháp ML phải khái quát hóa vượt ra ngoài dữ liệu đào tạo của nó, ví dụ, chuyển sang các trường hợp có kích thước khác nhau.

  Overall, there is a trade-off between scalability, expressivity, & generalization, which might conflict. In summary, key challenges are:

  1. ML methods that operate on graph data have to be invariant to node permutations. They should also exploit graphs' sparsity.
  2. Models should distinguish critical structural patterns in provided data while still scaling to large real-world instances.
  3. Side information in form of high-dimensional vectors attached to nodes & edges, i.e., modeling objectives & additional information, need to be considered.
  4. Models should be data efficient, i.e., they should ideally work without requiring large amounts of labeled data, & they should be transferable to out-of-sample or out-of-distribution instances.

– Nhìn chung, có sự đánh đổi giữa khả năng mở rộng, khả năng biểu đạt, & khái quát hóa, & những yếu tố này có thể xung đột. Tóm lại, những thách thức chính là:

1. Các phương pháp ML hoạt động trên dữ liệu đồ thị phải bất biến với các hoán vị nút. Chúng cũng nên khai thác tính thưa thớt của đồ thị.
2. Các mô hình nên phân biệt các mẫu cấu trúc quan trọng trong dữ liệu được cung cấp trong khi vẫn có thể mở rộng đến các trường hợp thực tế lớn.
3. Thông tin phụ dưới dạng các vectơ nhiều chiều được gắn vào các nút & cạnh, tức là các mục tiêu mô hình & thông tin bổ sung, cần được xem xét.
4. Các mô hình nên hiệu quả về dữ liệu, nghĩa là lý tưởng nhất là chúng nên hoạt động mà không cần lượng lớn dữ liệu được gắn nhãn, & chúng nên có thể chuyển giao cho các trường hợp ngoài mẫu hoặc ngoài phân phối.

- 1.2. How Do GNNs Address These Challenges? GNNs (Gilmer et al., 2017; Scarselli et al., 2009) have recently emerged ss ML architectures that partially address challenges above.

  Key idea underlying GNNs: compute a vectorial representation, e.g., a real vector, of each node in input graph by iteratively aggregating features of neighboring nodes. GNN is trained in an end-to-end fashion against a loss function, using (stochastic) 1st-order optimization techniques to adapt to given data distribution by parameterizing this aggregation step. Promise here: learned vector representation encodes crucial graph structures that help solve a CO problem more efficiently. GNNs are invariant & equivariant by design, i.e., they automatically exploit invariances or symmetries inherent to given instance or data distribution. Due to their local nature, by aggregating neighborhood information, GNNs naturally exploit sparsity, leading to more scalable models on sparse inputs. Moreover, although scalability is still an issue, they scale linearly with number of edges & employed parameters while taking multidimensional node & edge features into account (Gilmer et al., 2017), naturally exploiting cost & objective function information. However, data-efficiency question is still large open (Morris et al., 2021).

  – Ý tưởng chính nền tảng của GNN: tính toán biểu diễn vectơ, ví dụ: 1 vectơ thực, của mỗi nút trong đồ thị đầu vào bằng cách tổng hợp lặp lại các đặc điểm của các nút lân cận. GNN được huấn luyện theo kiểu đầu cuối-cuối dựa trên hàm mất mát, sử dụng các kỹ thuật tối ưu hóa bậc nhất (ngẫu nhiên) để thích ứng với phân phối dữ liệu cho trước bằng cách tham số hóa bước tổng hợp này. Lời hứa ở đây: biểu diễn vectơ đã học mã hóa các cấu trúc đồ thị quan trọng giúp giải quyết vấn đề CO hiệu quả hơn. GNN được thiết kế bất biến & tương đương, tức là chúng tự động khai thác các bất biến hoặc tính đối xứng vốn có trong phân phối dữ liệu hoặc trường hợp cho trước. Do tính chất cục bộ của chúng, bằng cách tổng hợp thông tin lân cận, GNN tự nhiên khai thác tính thưa thớt, dẫn đến các mô hình có khả năng mở rộng hơn trên các đầu vào thưa thớt. Hơn nữa, mặc dù khả năng mở rộng vẫn là 1 vấn đề, nhưng chúng mở rộng tuyến tính với số lượng cạnh & tham số được sử dụng trong khi tính đến các đặc điểm nút & cạnh đa chiều (Gilmer & cộng sự, 2017), tự nhiên khai thác thông tin về chi phí & hàm mục tiêu. Tuy nhiên, câu hỏi về hiệu quả dữ liệu vẫn còn bỏ ngỏ (Morris & cộng sự, 2021).

  Although GNNs have clear limitations, they have already proven useful in CO. In fact, they have already been applied in various settings, either to directly predict a solution or as an integrated component of an existing solver. Extensively investigate both of these aspects within our survey.

  – Mặc dù GNN có những hạn chế rõ ràng, nhưng chúng đã chứng minh được tính hữu ích trong CO. Trên thực tế, chúng đã được áp dụng trong nhiều bối cảnh khác nhau, hoặc để dự đoán trực tiếp 1 giải pháp, hoặc như 1 thành phần tích hợp của 1 bộ giải hiện có. Hãy nghiên cứu sâu rộng cả 2 khía cạnh này trong khảo sát của chúng tôi.

  Perhaps 1 of most widely publicized applications of GNNs in CO at time of writing is work by Mirhoseini et al. (2021), which studies chip placement. Aim: map nodes of a netlist (graph describing desired chip) onto a chip canvas (a bounded 2D space), optimizing final power, performance, & area. Authors observe this as a combinatorial problem & tackle it using reinforcement learning. Owning to graph structure of netlist, at core of representation learning pipeline is a GNN, which computes node features in a (permutation-)invariant way. This represents 1st chip placement approach that can quickly generalize to previously unseen netlists, generating optimized placements for Google's TPU accelerators (Jouppi et al., 2017). While this approach has received wide coverage in popular press, believe it only scratches surface of innovations that can be enabled by a careful synergy of GNNs & CO. Have designed our survey to facilitate future research in this emerging area.

  – Có lẽ 1 trong những ứng dụng được công bố rộng rãi nhất của GNN trong CO tại thời điểm viết bài này là công trình của Mirhoseini & cộng sự (2021), nghiên cứu về vị trí đặt chip. Mục tiêu: ánh xạ các nút của netlist (đồ thị mô tả chip mong muốn) lên 1 khung chip (không gian 2 chiều bị chặn), tối ưu hóa công suất cuối cùng, hiệu suất & diện tích. Các tác giả coi đây là 1 vấn đề tổ hợp & giải quyết nó bằng cách sử dụng học tăng cường. Nhờ cấu trúc đồ thị của netlist, cốt lõi của quy trình học biểu diễn là GNN, tính toán các đặc điểm của nút theo cách bất biến (hoán vị). Đây là phương pháp tiếp cận vị trí đặt chip đầu tiên có thể nhanh chóng khái quát hóa thành các netlist chưa từng thấy trước đây, tạo ra các vị trí tối ưu cho các bộ tăng tốc TPU của Google (Jouppi & cộng sự, 2017). Mặc dù phương pháp này đã được báo chí đưa tin rộng rãi, nhưng chúng tôi tin rằng nó chỉ mới khai thác bề mặt của những đổi mới có thể được kích hoạt bằng sự phối hợp cẩn thận giữa GNN & CO. Chúng tôi đã thiết kế khảo sát của mình để tạo điều kiện cho các nghiên cứu trong tương lai trong lĩnh vực mới nổi này.

- 1.3. Going Beyond Classical Algorithms. Previous discussion mainly dealt with ML approaches, especially GNNs, replacing & imitating classical combinatorial algorithms or parts of them, potentially adapting better to specific data distribution of naturally-occurring problem instances. However, classical algorithms heavily depend on human-made pre-processing or feature engineering by abstracting raw, real-world inputs, e.g., specifying underlying graph itself. Discrete graph input, forming

basis of most CO problems, is seldom directly induced by raw data, requiring costly & error-prone feature engineering. This might lead to biases that do not align with real world &, consequently, imprecise decisions. Such issues have been known have been known as early as 1950s in context of railways network analysis (Harris & Ross, 1955), but remained out of spotlight of theoretical CS that assumes problems are abstractified, to begin with.

– Vượt ra ngoài các thuật toán cổ điển. Các cuộc thảo luận trước đây chủ yếu đề cập đến các phương pháp ML, đặc biệt là GNN, thay thế & mô phỏng các thuật toán tổ hợp cổ điển hoặc 1 phần của chúng, có khả năng thích ứng tốt hơn với phân phối dữ liệu cụ thể của các trường hợp vấn đề xảy ra tự nhiên. Tuy nhiên, các thuật toán cổ điển phụ thuộc rất nhiều vào quá trình tiền xử lý hoặc kỹ thuật đặc trưng do con người tạo ra bằng cách trừu tượng hóa các đầu vào thô, thực tế, ví dụ: chỉ định chính đồ thị cơ bản. Đầu vào đồ thị rời rạc, tạo thành cơ sở của hầu hết các vấn đề CO, hiếm khi được tạo trực tiếp từ dữ liệu thô, đòi hỏi kỹ thuật đặc trưng dễ xảy ra lỗi & tốn kém. Điều này có thể dẫn đến các sai lệch không phù hợp với thế giới thực &, do đó, đưa ra các quyết định không chính xác. Những vấn đề như vậy đã được biết đến từ những năm 1950 trong bối cảnh phân tích mạng lưới đường sắt (Harris & Ross, 1955), nhưng vẫn nằm ngoài tầm chú ý của khoa học máy tính lý thuyết vốn giả định rằng các vấn đề đã được trừu tượng hóa ngay từ đầu.

In long-term, ML approaches can further enhance CO pipeline, from raw input processing to aiding in solving abstracted CO problems in an end-to-end fashion. Several viable approaches in this direction have been proposed recently, & survey them in detail, along with motivating examples, in Sect. 3.3.3.

– Về lâu dài, các phương pháp tiếp cận ML có thể cải thiện hơn nữa quy trình CO, từ xử lý dữ liệu đầu vào thô đến hỗ trợ giải quyết các vấn đề CO trừu tượng theo cách toàn diện. 1 số phương pháp khả thi theo hướng này đã được đề xuất gần đây, & xem xét chi tiết chúng, cùng với các ví dụ minh họa, trong Phần 3.3.3.

- 1.4. Present work. In this paper, give an overview of recent advances in using GNNs in context of CO, aiming at both CO & ML researchers. To this end, thoroughly introduce CO, ML regimes, & GNNs. Most importantly, give a comprehensive, structured overview of recent applications of GNNs in CO context. Discuss challenges arising from use of GNNs & future work. Our contributions can be summarized as follows:

  1. Provide a comprehensive, structured overview of application of GNNs to CO setting for both heuristic & exact algorithms.
  2. Survey recent progress in using GNN-based end-to-end algorithmic reasoners.
  3. Highlight shortcomings of GNNs in context of CO & provide guidelines & recommendations on how to tackle them.
  4. Provide a list of open research directions to stimulate future research.

  – Trong bài báo này, chúng tôi sẽ trình bày tổng quan về những tiến bộ gần đây trong việc sử dụng GNN trong bối cảnh CO, hướng đến cả các nhà nghiên cứu CO & ML. Để đạt được mục đích này, chúng tôi sẽ giới thiệu kỹ lưỡng về CO, các chế độ ML, & GNN. Quan trọng nhất, chúng tôi sẽ cung cấp 1 cái nhìn tổng quan toàn diện, có cấu trúc về các ứng dụng gần đây của GNN trong bối cảnh CO. Thảo luận về những thách thức phát sinh từ việc sử dụng GNN & các nghiên cứu trong tương lai. Những đóng góp của chúng tôi có thể được tóm tắt như sau:

  1. Cung cấp 1 cái nhìn tổng quan toàn diện, có cấu trúc về ứng dụng GNN vào thiết lập CO cho cả thuật toán heuristic & chính xác.
  2. Khảo sát những tiến bộ gần đây trong việc sử dụng các bộ suy luận thuật toán đầu cuối dựa trên GNN.
  3. Nêu bật những hạn chế của GNN trong bối cảnh CO & cung cấp các hướng dẫn & khuyến nghị về cách khắc phục chúng.
  4. Cung cấp danh sách các hướng nghiên cứu mở để thúc đẩy nghiên cứu trong tương lai.

  Believe: reaping benefits of GNNs for CO is a promising research direction. This survey is intended to provide required material for readers eager to discover this field. On other hand, highlight: this survey should not be considered as a recommendation that GNNs is best way to solve CO problems. There are clear limitations & fundamental challenges to tackle. Besides, results obtained are currently below what can be achieved by traditional CO solvers in most situations. Such limitations are discussed throughout paper & summarized in Sect. 4.

  – Tin tưởng: việc tận dụng lợi ích của GNN cho bài toán CO là 1 hướng nghiên cứu đầy hứa hẹn. Khảo sát này nhằm mục đích cung cấp tài liệu cần thiết cho những độc giả mong muốn khám phá lĩnh vực này. Mặt khác, xin lưu ý: khảo sát này không nên được coi là 1 khuyến nghị rằng GNN là giải pháp tốt nhất để giải quyết các vấn đề CO. Có những hạn chế rõ ràng & những thách thức cơ bản cần giải quyết. Hơn nữa, kết quả thu được hiện nay còn thấp hơn so với những gì các bộ giải CO truyền thống có thể đạt được trong hầu hết các trường hợp. Những hạn chế này được thảo luận trong toàn bộ bài báo & tóm tắt trong Phần 4.

- 1.5. Related work. Following briefly reviews key papers & survey efforts involving GNNs & ML for CO.

  GNNs. Graph neural networks (Gilmer et al., 2017; Scarselli et al., 2009) have recently (re-)emerged as leading ML method for graph-structured inputs. Notable instances of this architecture include, e.g., Duvenaud et al. (2015); Hamilton et al. (2017); Veličković et al. (2018), & spectral approaches proposed by, e.g., Bruna et al. (2014); Defferrard et al. (2016); Kipf &Welling (2017); Monti et al. (2017) – all of which descend from early work of Kireev (1995); Merkwirth &Lengauer (2005); Scarselli et al. (2009); Sperduti &Starita (1997). Aligned with field's recent rise in popularity, many surveys exist on recent advances in GNN techniques. Some of most recent ones include Chami et al. (2020); Wu et al. (2019); Zhou et al. (2020).

  Continuous Formulations. Discrete nature of CO problems makes standard continuous optimization tools unavailable, e.g. 1st- & 2nd-order gradient methods. However, many problems admit alternative reformulations as nonconvex continuous optimization problems over graphs. Such problems include graph partitioning, maximum cut, minimum vertex cover, maximum independent set, & maximum clique problems. Some early work at intersection of ML & combinatorial optimization involves reinterpreting these continuous optimization problems as energy-based training of Hopfield neural networks or self-organizing maps, e.g. in work of Hopfield & Tank (1985), Durbin & Willshaw (1987), Ramanujam &Sadayappan (1995) &

Gold et al. (1996). Although not using GNNs, these works use graphs as a central object. They can be seen as foreshadowing various GNN-based differentiable proxy loss approaches summarized in Sect. 3.1.1.

– **Công thức liên tục.** Bản chất rời rạc của các vấn đề CO khiến các công cụ tối ưu hóa liên tục tiêu chuẩn không khả dụng, ví dụ: phương pháp gradient bậc 1 & 2. Tuy nhiên, nhiều vấn đề cho phép các công thức thay thế như các vấn đề tối ưu hóa liên tục không lồi trên đồ thị. Các vấn đề như vậy bao gồm phân vùng đồ thị, cắt cực đại, phủ đỉnh cực tiểu, tập độc lập cực đại, & các vấn đề clique cực đại. 1 số công trình ban đầu tại giao điểm của ML & tối ưu hóa tổ hợp liên quan đến việc diễn giải lại các vấn đề tối ưu hóa liên tục này dưới dạng huấn luyện dựa trên năng lượng của mạng nơ-ron Hopfield hoặc bản đồ tự tổ chức, ví dụ: trong công trình của Hopfield & Tank (1985), Durbin & Willshaw (1987), Ramanujam & Sadayappan (1995) & Gold et al. (1996). Mặc dù không sử dụng GNN, các công trình này sử dụng đồ thị làm đối tượng trung tâm. Chúng có thể được coi là điềm báo trước các phương pháp tiếp cận mất mát proxy khả vi dựa trên GNN khác nhau được tóm tắt trong Phần 3.1.1.

Surveys. Seminal survey of Smith (1999) centers around use of popular neural network architectures of time, namely Hopfield networks & self-organizing maps, as a basis for combinatorial heuristics, as described in prev sect. Worth noting: such architectures were mostly used for a single instance at a time rather than being trained over a set of training instances; this may explain their limited success at time. Bengio et al. (2021) give a high-level overview of ML methods for CO, with no special focus on graph-structured input, while Lodi & Zarpellon (2017) focus on ML for branching in context of mixed-integer programming. Concurrently to our work, Kotary et al. (2021) have categorized various approaches for ML in CO, focusing primarily on end-to-end learning setups & paradigms, making representation learning – & GNNs in particular – a secondary topic. Moreover, surveys by Mazyavkina et al. (2021); Yang & Whinston (2020) focus on using reinforcement learning for CO. Survey of Vesselinova et al. (2020) deals with ML for network problems arising in telecommunications, focusing on non-exact methods & not including recent progress. Finally, Lamb et al. (2020) give a high-level overview of application of GNNs in various reasoning tasks, missing out on most recent developments, e.g., algorithmic reasoning direction we study in detail here.

– **Khảo sát.** Khảo sát quan trọng của Smith (1999) tập trung vào việc sử dụng các kiến trúc mạng nơ-ron phổ biến theo thời gian, cụ thể là mạng Hopfield & bản đồ tự tổ chức, làm cơ sở cho các phương pháp tìm kiếm kết hợp, như được mô tả trong phần trước. Điều đáng chú ý: các kiến trúc như vậy chủ yếu được sử dụng cho 1 trường hợp duy nhất tại 1 thời điểm thay vì được đào tạo qua 1 tập hợp các trường hợp đào tạo; điều này có thể giải thích thành công hạn chế của chúng theo thời gian. Bengio & cộng sự (2021) đưa ra tổng quan cấp cao về các phương pháp ML cho CO, không tập trung đặc biệt vào đầu vào có cấu trúc đồ thị, trong khi Lodi & Zarpellon (2017) tập trung vào ML để phân nhánh trong bối cảnh lập trình số nguyên hỗn hợp. Đồng thời với công trình của chúng tôi, Kotary & cộng sự (2021) đã phân loại các phương pháp tiếp cận khác nhau cho ML trong CO, chủ yếu tập trung vào các thiết lập học tập đầu cuối & các mô hình, đưa việc học biểu diễn – & GNN nói riêng – thành 1 chủ đề thứ yếu. Hơn nữa, các khảo sát của Mazyavkina & cộng sự (2021); Yang & Whinston (2020) tập trung vào việc sử dụng học tăng cường cho CO. Khảo sát của Vesselinova & cộng sự (2020) đề cập đến ML cho các vấn đề mạng phát sinh trong viễn thông, tập trung vào các phương pháp không chính xác & không bao gồm những tiến bộ gần đây. Cuối cùng, Lamb & cộng sự (2020) đưa ra tổng quan cấp cao về ứng dụng của GNN trong các tác vụ suy luận khác nhau, bỏ qua hầu hết các phát triển gần đây, ví dụ như hướng suy luận thuật toán mà chúng tôi sẽ nghiên cứu chi tiết tại đây.

- 1.6. Outline. Start by giving necessary background on CO & relevant optimization frameworks, ML, & GNNs; see Sect. 2. In Sect. 3, review recent research using GNNs in CO context. Specifically, in Sect. 3.1, survey works aiming at finding primal solutions, i.e., high-quality feasible solutions to CO problems, while Sect. 3.2 gives an overview of works aiming at enhancing dual methods, i.e., proving optimality of solutions. Going beyond that, Sect. 3.3 reviews recent research trying to facilitate algorithmic reasoning behavior in GNNs, as well as applying GNNs as raw-input combinatorial optimizers. Finally, Sect. 4 discusses limits of current approaches & offers a list of research directions intending to stimulate future research.

  – Bắt đầu bằng cách cung cấp kiến thức nền tảng cần thiết về CO & các khuôn khổ tối ưu hóa liên quan, ML, & GNN; xem Mục 2. Trong Mục 3, hãy xem xét các nghiên cứu gần đây sử dụng GNN trong bối cảnh CO. Cụ thể, trong Mục 3.1, các công trình khảo sát nhằm tìm kiếm các giải pháp nguyên thủy, tức là các giải pháp khả thi chất lượng cao cho các bài toán CO, trong khi Mục 3.2 cung cấp tổng quan về các công trình nhằm cải thiện các phương pháp đối ngẫu, tức là chứng minh tính tối ưu của các giải pháp. Ngoài ra, Mục 3.3 xem xét các nghiên cứu gần đây cố gắng tạo điều kiện cho hành vi suy luận thuật toán trong GNN, cũng như ứng dụng GNN làm bộ tối ưu hóa tổ hợp đầu vào thô. Cuối cùng, Mục 4 thảo luận về các giới hạn của các phương pháp tiếp cận hiện tại & đưa ra danh sách các hướng nghiên cứu nhằm kích thích các nghiên cứu trong tương lai.

- 2. Preliminaries. Here introduce notation & give necessary formal background on combinatorial optimization, different ML regimes, & GNNs.

  - 2.1. Notation. Let $[n] = \{1, \dots, n\} \subset \mathbb{N}$ for $n \geq 1$, & let $\{\{\dots\}\}$ denote a multiset. For a (finite) set S, denote its power set as 2^S. A graph G is a pair (V, E) with a finite set of nodes V & a set of edges $E \subseteq V \times V$. Denote set of nodes & set of edges of G by $V(G), E(G)$, resp. A labeled graph G is a triplet (V, E, l) with a label function $l : V(G) \cup E(G) \to \Sigma$, where Σ is some finite alphabet. Then $l(x)$ is label of x, for $x \in V(G) \cup E(G)$. Note x here can be either a node or an edge. Neighborhood of v in $V(G)$ is denoted by $N(v) = \{u \in V(G); (v, u) \in E(G)\}$. A tree is a connected graph without cycles.

    – Cho $[n] = \{1, \dots, n\} \subset \mathbb{N}$ với $n \geq 1$, & cho $\{\{\dots\}\}$ biểu thị 1 đa tập. Với 1 tập (hữu hạn) S, hãy ký hiệu tập lũy thừa của nó là 2^S. 1 đồ thị G là 1 cặp (V, E) với 1 tập hữu hạn các nút V & 1 tập các cạnh $E \subseteq V \times V$. Ký hiệu tập các nút & tập các cạnh của G theo $V(G), E(G)$, tương ứng. 1 đồ thị có nhãn G là 1 bộ ba (V, E, l) với hàm nhãn $l : V(G) \cup E(G) \to \Sigma$,

trong đó Σ là 1 bảng chữ cái hữu hạn. Khi đó $l(x)$ là nhãn của x, với $x \in V(G) \cup E(G)$. Lưu ý x ở đây có thể là 1 nút hoặc 1 cạnh. Lân cận của v trong $V(G)$ được ký hiệu là $N(v) = \{u \in V(G); (v,u) \in E(G)\}$. Cây là 1 đồ thị liên thông không có chu trình.

Say 2 graphs G, H are isomorphic if there exists an edge-preserving bijection $\varphi : V(G) \to V(H)$, i.e., $(u,v) \in E(G) \Leftrightarrow (\varphi(u), \varphi(v)) \in E(H)$. For labeled graphs, further require $l(v) = l(\varphi(v))$ for $v \in V(G)$ & $l((u,v)) = l((\varphi(u), \varphi(v)))$ for $(u,v) \in E(G)$.

– Giả sử 2 đồ thị G, H là đẳng cấu nếu tồn tại 1 song ánh bảo toàn cạnh $\varphi : V(G) \to V(H)$, tức là $(u,v) \in E(G) \Leftrightarrow (\varphi(u), \varphi(v)) \in E(H)$. Đối với đồ thị có nhãn, cần thêm $l(v) = l(\varphi(v))$ với $v \in V(G)$ & $l((u,v)) = l((\varphi(u), \varphi(v)))$ với $(u,v) \in E(G)$.

- 2.2. Combinatorial optimization. CO deals with problems that involve optimizing a cost (or objective) function by selecting a subset from a finite set, with latter encoding constraints on solution space. Formally, define an instance of a combinatorial optimization problem as follows.

  – CO xử lý các bài toán liên quan đến việc tối ưu hóa hàm chi phí (hoặc hàm mục tiêu) bằng cách chọn 1 tập con từ 1 tập hữu hạn, sau đó mã hóa các ràng buộc trên không gian nghiệm. Định nghĩa 1 cách hình thức 1 bài toán tối ưu hóa tổ hợp như sau.

  Definition 8 (Combinatorial optimization instance)**.** *An instance of a* combinatorial optimization problem *is a tuple* (Ω, F, w)*, where* Ω *is a finite set,* $F \subseteq 2^\Omega$ *is set of feasible solutions,* $c : 2^\Omega \to \mathbb{R}$ *is a* cost function *with* $c(S) = \sum_{\omega \in S} w(\omega)$ *for* $S \in F$*.*

  Consequently, CO deals with selecting an element S^* (optimal solution) in F that minimizes c over feasible set F. [W.l.o.g. choose minimization instead of maximization.] Corresponding decision problem asks if there exists an element in feasible set s.t. its cost is $\leq$ a given value, i.e., whether there exists $S \in F$ s.t. $c(S) \leq k$, i.e., require a Yes/No answer.

  – Do đó, CO xử lý việc lựa chọn 1 phần tử S^* (giải pháp tối ưu) trong F sao cho tối thiểu hóa c trên tập hợp khả thi F. [W.l.o.g. chọn tối thiểu hóa thay vì tối đa hóa.] Bài toán quyết định tương ứng hỏi liệu có tồn tại 1 phần tử trong tập hợp khả thi s.t. hay không, chi phí của nó là ≤ 1 giá trị cho trước, tức là liệu có tồn tại $S \in F$ s.t. $c(S) \leq k$ hay không, tức là yêu cầu câu trả lời Có/Không.

  TSP is a well-known CO problem aiming at finding a cycle along edges of a graph with minimal cost that visits each node exactly once; see Fig. 1: A complete graph with edge labels (blue & red) & its optimal solution for TSP (in gree). Blue edges have a cost of 1 & red edges a cost of 2. for an illustration of an instance of TSP problem & its optimal solution. Corresponding decision problem asks whether a cycle exists along edges of a graph with cost $\leq k$ that visits each node exactly once.

  – TSP là 1 bài toán CO nổi tiếng, nhằm mục đích tìm 1 chu trình dọc theo các cạnh của 1 đồ thị có chi phí tối thiểu, đi qua mỗi nút đúng 1 lần; xem Hình 1: 1 đồ thị hoàn chỉnh với các nhãn cạnh (xanh lam & đỏ) & giải pháp tối ưu cho TSP (bằng màu xanh lục). Các cạnh xanh lam có chi phí là 1 & các cạnh đỏ có chi phí là 2. để minh họa 1 ví dụ về bài toán TSP & giải pháp tối ưu của nó. Bài toán quyết định tương ứng hỏi liệu có tồn tại 1 chu trình dọc theo các cạnh của 1 đồ thị có chi phí $\leq k$ đi qua mỗi nút đúng 1 lần hay không.

  Example 1 (Traveling Salesperson Problem)**.**

  Input. *A complete directed graph* G*, i.e.,* $E(G) = \{(u,v); u, v \in V(G)\}$*, with edge costs* $w : E(G) \to \mathbb{R}$*.*

  Output. *A permutation of nodes* $\sigma : \{0, \ldots, n-1\} \to V$ *s.t.* $\sum_{i=0}^{n-1} w((\sigma(i), \sigma((i+1) \mod n)))$ *is minimal over all permutations, where* $n = |V|$*.*

  Đầu vào. *1 đồ thị có hướng đầy đủ* G*, tức là* $E(G) = \{(u,v); u, v \in V(G)\}$*, với chi phí cạnh* $w : E(G) \to \mathbb{R}$*.*

  Đầu ra. *1 hoán vị của các nút* $\sigma : \{0, \ldots, n-1\} \to V$ *s.t.* $\sum_{i=0}^{n-1} w((\sigma(i), \sigma((i+1) \mod n)))$ *là tối thiểu trên tất cả các hoán vị, trong đó* $n = |V|$*.*

  Due to their discrete nature, many classes or sets of combinatorial decision problems arising in practice, e.g., TSP or other vehicle routing problems, are NP-hard (Korte & Vygen, 2012), & hence likely intractable in worst-case sense. However, instances are routinely solved in practice by formulating them as integer linear optimization problems or integer linear programs (ILPs), constrained problems, or as satisfiability problems (SAT) & utilizing well-engineered algorithms (& associated solvers) for these problems, e.g., branch-&-cut algorithms in case of ILPs.

  – Do tính chất rời rạc của chúng, nhiều lớp hoặc tập hợp các bài toán quyết định tổ hợp phát sinh trong thực tế, ví dụ như TSP hoặc các bài toán định tuyến phương tiện khác, là NP-khó (Korte & Vygen, 2012), & do đó có khả năng khó giải quyết trong trường hợp xấu nhất. Tuy nhiên, các trường hợp này thường được giải quyết trong thực tế bằng cách xây dựng chúng dưới dạng các bài toán tối ưu tuyến tính số nguyên hoặc chương trình tuyến tính số nguyên (ILP), các bài toán ràng buộc, hoặc các bài toán khả năng thỏa mãn (SAT) & sử dụng các thuật toán được thiết kế tốt (& các bộ giải liên quan) cho các bài toán này, ví dụ như thuật toán cắt nhánh trong trường hợp ILP.

- 2.3. General Optimization Frameworks: ILPs, SAT, & Constrained Problems. Describe common modeling & algorithmic frameworks for CO problems in following. More precisely, next 3 sects describe modeling approaches: integer programming, SAT, & constraint satisfaction/optimization. Finally, Sect. 2.3.4 partitions algorithmic frameworks into 3 categories. Note: this sect is not an exhaustive list of optimization approaches but serves as an introduction to main frameworks.

  – . Khung Tối ưu hóa Chung: ILP, SAT, & Bài toán Ràng buộc. Mô tả các khung thuật toán & mô hình hóa phổ biến cho các bài toán CO sau đây. Chính xác hơn, 3 phần tiếp theo mô tả các phương pháp mô hình hóa: lập trình số nguyên, SAT, & tối ưu hóa thỏa mãn ràng buộc. Cuối cùng, Phần 2.3.4 phân chia các khung thuật toán thành 3 loại. Lưu ý: phần này không phải là danh sách đầy đủ các phương pháp tối ưu hóa mà chỉ là phần giới thiệu về các khung chính.

∗ 2.3.1. Integer linear programs & mixed-integer programs. 1st, define a linear program or linear optimization problem. A linear program aims at optimizing a linear cost function over a feasible set described as intersection of finitely many half-spaces, i.e., a polyhedron. Formally, define an instance of a linear program as follows.
– Chương trình tuyến tính nguyên & Chương trình số nguyên hỗn hợp. Trước tiên, hãy định nghĩa 1 chương trình tuyến tính hoặc bài toán tối ưu tuyến tính. 1 chương trình tuyến tính nhằm mục đích tối ưu hóa 1 hàm chi phí tuyến tính trên 1 tập hợp khả thi được mô tả là giao của hữu hạn các nửa không gian, tức là 1 đa diện. Định nghĩa 1 cách hình thức 1 thể hiện của chương trình tuyến tính như sau.

Definition 9 (Linear programming instance)**.** *An instance of a* linear program *(LP) is a tuple* (A, b, c)*, where* A *is a matrix in* $\mathbb{R}^{m\times n}$*,* b, c *are vectors in* $\mathbb{R}^m, \mathbb{R}^n$*, resp.*

Định nghĩa 3 (Thể hiện lập trình tuyến tính)**.** *1 thể hiện của* chương trình tuyến tính *(LP) là 1 bộ* (A, b, c)*, trong đó* A *là 1 ma trận trong* $\mathbb{R}^{m\times n}$*,* b, c *là các vectơ trong* $\mathbb{R}^m, \mathbb{R}^n$*, tương ứng.*

Associated optimization problem asks to minimize a linear objective over a polyhedron. [In above def, assumed: LP is feasible, i.e., $X \neq \emptyset$, & a finite minimum value exists. In what follows, assume: both conditions are always fulfilled.] I.e., aim at finding a vector $x \in \mathbb{R}^n$ that minimizes $c^\top x$ over feasible set

$$X = \{x \in \mathbb{R}^n; A_j x \leq b_j \text{ for } j \in [m],\ x_i \geq 0 \text{ for } i \in [n]\}.$$

In practice, LPs are solved using Simplex method or polynomial-time interior-point methods (Bertsimas & Tsitsiklis, 1997). Due to their continuous nature, LPs cannot encode feasible set of a CO problem. Hence, extend LPs by adding integrality constraints, i.e., requiring: value assigned to each variable is an integer. Consequently, aim to find vector $x \in \mathbb{Z}^n$ that minimizes $c^\top x$ over feasible set

$$X = \{x \in \mathbb{Z}^n; A_j x \leq b_j \text{ for } j \in [m],\ x_i \geq 0,\ x_i \in \mathbb{Z} \text{ for } i \in [n]\}.$$

Such integer linear optimization problems are solved by tree search algorithms, e.g., branch-&-bound algorithms. By dropping integrality constraints, again obtain an instance of an LP, which recall relaxation. Solving LP relaxation of an ILP provides a valid lower bound on optimal solution of problem, i.e., an optimistic approximation, & quality of such an approximation is largely responsible of effectiveness of search scheme.
– Bài toán tối ưu hóa liên quan yêu cầu tối thiểu hóa 1 mục tiêu tuyến tính trên 1 đa diện. [Trong định nghĩa trên, giả sử: LP khả thi, tức là $X \neq \emptyset$, & tồn tại 1 giá trị cực tiểu hữu hạn. Trong phần tiếp theo, giả sử: cả 2 điều kiện luôn được thỏa mãn.] Tức là, mục tiêu là tìm 1 vectơ $x \in \mathbb{R}^n$ tối thiểu hóa $c^\top x$ trên tập khả thi

$$X = \{x \in \mathbb{R}^n; A_j x \leq b_j \text{ với } j \in [m],\ x_i \geq 0 \text{ với } i \in [n]\}.$$

Trong thực tế, LP được giải bằng phương pháp Simplex hoặc phương pháp điểm trong thời gian đa thức (Bertsimas & Tsitsiklis, 1997). Do tính chất liên tục của chúng, LP không thể mã hóa tập khả thi của 1 bài toán CO. Do đó, hãy mở rộng LP bằng cách thêm các ràng buộc tích phân, tức là yêu cầu: giá trị được gán cho mỗi biến là 1 số nguyên. Do đó, hãy tìm vectơ $x \in \mathbb{Z}^n$ sao cho $c^\top x$ là nhỏ nhất trên tập hợp khả thi

$$X = \{x \in \mathbb{Z}^n; A_j x \leq b_j \text{ với } j \in [m],\ x_i \geq 0,\ x_i \in \mathbb{Z} \text{ với } i \in [n]\}.$$

Các bài toán tối ưu tuyến tính số nguyên như vậy được giải bằng các thuật toán tìm kiếm cây, ví dụ như các thuật toán nhánh-&-bound. Bằng cách loại bỏ các ràng buộc tích phân, 1 lần nữa thu được 1 thể hiện của LP, gợi nhớ đến sự thư giãn. Giải quyết việc nới lỏng LP của ILP cung cấp 1 giới hạn dưới hợp lệ cho giải pháp tối ưu của vấn đề, tức là 1 phép tính gần đúng lạc quan, & chất lượng của phép tính gần đúng như vậy phần lớn chịu trách nhiệm cho hiệu quả của lược đồ tìm kiếm.

Example 2. *Provide an ILP that encodes all feasible solutions of TSP & selects optimal one due to objective function. Essentially, it encodes order of nodes or cities within its variables. Thereto, let*

$$x_{ij} = \begin{cases} 1 & \text{if the cycle goes from city } i \text{ to city } j, \\ 0 & \text{otherwise,} \end{cases}$$

& let $w_{ij} > 0$ *be cost or distance of traveling from city* i *to city* j*,* $i \neq j$*. Then TSP can be written as following ILP [Technically, presented TSP model is for asymmetric version, where costs* w_{ij}, w_{ji} *might be different. Such a TSP version is represented in a directed graph. Instead, version in Fig. 1 is symmetric, i.e.,* $w_{ij} = w_{ji}$*, & it is represented on an undirected graph.]*

$$\min \sum_{i=1}^{n} \sum_{j=1, j\in i}^{n} w_{ij} x_{ij} \text{ subject to } \sum_{i=1, i\neq j}^{n} x_{ij} = 1,\ \forall j \in [n],\ \sum_{j=1,\ j\neq i}^{n} x_{ij} = 1,\ \forall i \in [n],\ \sum_{i\in Q} \sum_{j\notin Q} x_{ij} \geq 1,\ \forall Q \subsetneq [n],\ |Q| \geq 2.$$

1st 2 constraints encode that each city should have exactly 1 in-going & out-going edge, resp. Last constraint ensures: all cities are within same tour, i.e., no sub-tours exist (thus, returned solution is not a collection of smaller tours).
– Hãy cung cấp 1 ILP mã hóa tất cả các giải pháp khả thi của TSP & chọn ra giải pháp tối ưu dựa trên hàm mục tiêu. Về cơ bản, nó mã hóa thứ tự các nút hoặc thành phố trong các biến của nó. Do đó, hãy cho

$$x_{ij} = \begin{cases} 1 & \text{nếu chu trình đi từ thành phố } i \text{ đến thành phố } j, \\ 0 & \text{nếu ngược lại,} \end{cases}$$

& cho $w_{ij} > 0$ là chi phí hoặc khoảng cách di chuyển từ thành phố i đến thành phố j, $i \neq j$. Khi đó, TSP có thể được viết dưới dạng ILP sau [Về mặt kỹ thuật, mô hình TSP được trình bày dành cho phiên bản bất đối xứng, trong đó chi phí w_{ij}, w_{ji} có thể khác nhau. Phiên bản TSP như vậy được biểu diễn dưới dạng đồ thị có hướng. Thay vào đó, phiên bản trong Hình 1 là đối xứng, tức là $w_{ij} = w_{ji}$, & nó được biểu diễn trên 1 đồ thị vô hướng.]

$$\min \sum_{i=1}^{n} \sum_{j=1, j\in i}^{n} w_{ij}x_{ij} \text{ tùy thuộc vào } \sum_{i=1, i\neq j}^{n} x_{ij} = 1,\ \forall j \in [n],\ \sum_{j=1,\ j\neq i}^{n} x_{ij} = 1,\ \forall i \in [n],\ \sum_{i\in Q}\sum_{j\notin Q} x_{ij} \geq 1,\ \forall Q \subsetneq [n],\ |Q| \geq 2.$$

2 ràng buộc đầu tiên mã hóa rằng mỗi thành phố phải có đúng 1 cạnh vào & ra, tương ứng. Ràng buộc cuối cùng đảm bảo: tất cả các thành phố đều nằm trong cùng 1 hành trình, tức là không có hành trình con nào tồn tại (do đó, nghiệm trả về không phải là tập hợp các hành trình nhỏ hơn).

In practice, one often faces problems consisting of integer & continuous variables. These are commonly known as mixed-integer programs (MIPs). Formally, given $p \in \mathbb{N}^\star$, MIPs aim at finding a vector $x \in \mathbb{R}^n$ that minimizes $c^\top x$ over feasible set

$$X = \{x \in \mathbb{R}^n; A_j x \leq b_j \text{ for } j \in [m],\ x_i \geq 0 \text{ for } i \in [n],\ x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}.$$

Here n is number of variables we optimize, out of which p must be integers.

– Trong thực tế, người ta thường gặp phải các bài toán bao gồm các biến số nguyên & biến liên tục. Chúng thường được gọi là các chương trình số nguyên hỗn hợp (MIP). Về mặt hình thức, với $p \in \mathbb{N}^\star$, MIP hướng đến việc tìm 1 vectơ $x \in \mathbb{R}^n$ sao cho tối thiểu hóa $c^\top x$ trên 1 tập hợp khả thi

$$X = \{x \in \mathbb{R}^n; A_j x \leq b_j \text{ với } j \in [m],\ x_i \geq 0 \text{ với } i \in [n],\ x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}.$$

Ở đây n là số biến chúng ta cần tối ưu hóa, trong đó p phải là số nguyên.

∗ 2.3.2. SAT. Boolean satisfiability problem (SAT) asks, given a Boolean formula or propositional logic formula, if there exists a variable assignment (assign true or false to variables) s.t. formula evaluates to true. Hence, formally we can define it as follows.

Definition 10 (SAT)**.**

Input. *A propositional logic formula φ with variable set V.*

Output. *Yes if there exists a variable assignment $A : V \to \{true, false\}$ s.t. formula φ evaluates to true; No otherwise.*

Đầu vào. *1 công thức logic mệnh đề φ với tập biến V.*

Đầu ra. *Có nếu tồn tại phép gán biến $A : V \to \{true, false\}$ thì công thức φ được đánh giá là đúng; Nếu không thì không.*

SAT problem was 1st one to be shown to be NP-complete (Cook, 1971). However, modern solvers routinely solve industrial-scale instances in practice (Prasad et al., 2005). Despite simplicity of its formalization, SAT has many practical applications, e.g. hardware verification (Clarke et al., 2003; Gupta et al., 2006), configuration management (Mancinelli et al., 2006; Tucker et al., 2007), or planning (Behnke et al., 2018). A realistic case study of SAT is illustrated in:

– Bài toán SAT là bài toán đầu tiên được chứng minh là NP-complete (Cook, 1971). Tuy nhiên, các trình giải hiện đại thường xuyên giải quyết các trường hợp quy mô công nghiệp trong thực tế (Prasad & cộng sự, 2005). Mặc dù hình thức hóa đơn giản, SAT có nhiều ứng dụng thực tế, ví dụ như xác minh phần cứng (Clarke & cộng sự, 2003; Gupta & cộng sự, 2006), quản lý cấu hình (Mancinelli & cộng sự, 2006; Tucker & cộng sự, 2007), hoặc lập kế hoạch (Behnke & cộng sự, 2018). 1 nghiên cứu điển hình thực tế về SAT được minh họa trong:

Example 3. *Consider problem of installing a new (software) package P on a system where installation is subject to dependency & conflict constraints. Goal: determine which packages must be installed on system s.t. package P is installed in system, dependencies of all installed packages are satisfied, & there are no conflicts among installed packages. This problem can be conveniently modeled as an SAT problem (Tucker et al., 2007). Formally, [...] NQBH: seem relevant to XOR bitwise operations.*

– Xét bài toán cài đặt 1 gói (phần mềm) mới P trên 1 hệ thống, trong đó quá trình cài đặt phải tuân theo các ràng buộc phụ thuộc & xung đột. Mục tiêu: xác định những gói nào phải được cài đặt trên hệ thống. Ví dụ: gói P được cài đặt trong hệ thống, các ràng buộc phụ thuộc của tất cả các gói đã cài đặt đều được đáp ứng, & không có xung đột giữa các gói đã cài đặt. Bài toán này có thể được mô hình hóa 1 cách thuận tiện như 1 bài toán SAT (Tucker & cộng sự, 2007). Về mặt hình thức, [...] NQBH: dường như liên quan đến các phép toán XOR bitwise.

A natural extension of SAT is maximum satisfiability system (MaxSAT), which aims to determine maximum number of clauses, of a given Boolean formula in conjunctive normal form that can be evaluated to true by assigning truth values to variables.

– 1 phần mở rộng tự nhiên của SAT là hệ thống thỏa mãn tối đa (MaxSAT), nhằm mục đích xác định số lượng mệnh đề tối đa của 1 công thức Boolean nhất định ở dạng chuẩn kết hợp có thể được đánh giá là đúng bằng cách gán giá trị chân lý cho các biến.

∗ 2.3.3. Constraint satisfaction & constraint optimization problems. This sect presents both constraint satisfaction problems & constraint optimization problems, most generic way to formalize CO problems. Formally, an instance of a constraint satisfaction problem is defined as follows.

– Bài toán thỏa mãn ràng buộc & bài toán tối ưu hóa ràng buộc. Phần này trình bày cả bài toán thỏa mãn ràng buộc & bài toán tối ưu hóa ràng buộc, cách tổng quát nhất để hình thức hóa các bài toán CO. Về mặt hình thức, 1 ví dụ về bài toán thỏa mãn ràng buộc được định nghĩa như sau.

Definition 11 (Constraint satisfaction problem instance)**.** *An instance of a constraint satisfaction problem (CSP) is a tuple $(X, D(X), C)$, where X is set of variables, $D(X)$ is set of domains of variables, & C: set of constraints that restrict assignments of values to variables. A solution is an assignment of values from D to X that satisfies all constraints of C.*

Định nghĩa 4 (Ví dụ về vấn đề thỏa mãn ràng buộc)**.** *1 ví dụ về bài toán thỏa mãn ràng buộc (CSP) là 1 bộ $(X, D(X), C)$, trong đó X là tập hợp các biến, $D(X)$ là tập hợp các miền xác định của các biến, & C: tập hợp các ràng buộc hạn chế việc gán giá trị cho các biến. 1 giải pháp là việc gán các giá trị từ D cho X thỏa mãn mọi ràng buộc của C.*

A natural extension of CSPs are constrained optimization problems, i.e., CSPs that also have an objective function. Goal becomes finding a feasible assignment that minimizes objective function. Main difference with previous optimization frameworks: constrained optimization problems do not require underlying assumptions on variables, constraints, & objective functions. Unlike MIPs, nonlinear objectives & constraints are applied within this framework. E.g., a TSP model:

Example 4. *Given a configuration with n cities & a weight matrix $w \in \mathbb{R}^{n \times n}$, TSP can be modeled using n variables x_i over domain $D(x_i)$: $[n]$. Variable x_i indicates ith city to be visited. Objective function & constraints read as*

$$\min w_{x_n, x_1} + \sum_{i=1}^{n-1} w_{x_i, x_{i+1}} \text{ subject to } \text{allDiffrent}(x_1, \ldots, x_n),$$

where $allDifferent(X)$ enforces: each variable from X takes a different value (Régin, 1994), & entries of weight matrix w are indexed using variables. This model forces each city to have another city as a successor & sums up distances between each pair of consecutive cities along cycle.

– Với cấu hình có n thành phố & ma trận trọng số $w \in \mathbb{R}^{n \times n}$, TSP có thể được mô hình hóa bằng n biến x_i trên miền $D(x_i)$: $[n]$. Biến x_i biểu thị thành phố thứ i cần ghé thăm. Hàm mục tiêu & ràng buộc được đọc là

$$\min w_{x_n, x_1} + \sum_{i=1}^{n-1} w_{x_i, x_{i+1}} \text{ tuân theo } \text{allDiffrent}(x_1, \ldots, x_n),$$

trong đó $allDifferent(X)$ áp dụng: mỗi biến từ X nhận 1 giá trị khác nhau (Régin, 1994), & các mục của ma trận trọng số w được lập chỉ mục bằng các biến. Mô hình này buộc mỗi thành phố phải có 1 thành phố khác kế nhiệm & tổng hợp khoảng cách giữa mỗi cặp thành phố liên tiếp theo chu kỳ.

Constrained problems can model arbitrary constraints & objective functions. This generality makes it possible to use general-purpose solving methods e.g. local search or constraint programming. In addition to their convenience of modeling side, high-level constraints generally referred to as global constraints are also useful on solving side (Régin, 2004). They enable design of efficient algorithms dedicated to pruning search space. Leveraging pruning ability of global constraints is a fundamental component of a constraint programming solver as explained below.

– Các bài toán ràng buộc có thể mô hình hóa các ràng buộc & hàm mục tiêu tùy ý. Tính tổng quát này cho phép sử dụng các phương pháp giải đa năng, ví dụ như tìm kiếm cục bộ hoặc lập trình ràng buộc. Bên cạnh sự tiện lợi về mặt mô hình hóa, các ràng buộc cấp cao, thường được gọi là ràng buộc toàn cục, cũng hữu ích trong việc giải quyết bài toán (Régin, 2004). Chúng cho phép thiết kế các thuật toán hiệu quả chuyên biệt cho việc cắt tỉa không gian tìm kiếm. Việc tận dụng khả năng cắt tỉa của các ràng buộc toàn cục là 1 thành phần cơ bản của 1 bộ giải lập trình ràng buộc như được giải thích dưới đây.

∗ 2.3.4. Solving CO problems. Major algorithmic frameworks – whose components & tasks have been recently considered through GNNs lens – will be discussed when necessary in core of survey. However, in this sect, briefly distinguish 3 algorithmic categories.

– Giải quyết các vấn đề CO. Các khuôn khổ thuật toán chính – với các thành phần & nhiệm vụ gần đây đã được xem xét dưới góc nhìn của GNN – sẽ được thảo luận khi cần thiết trong phần cốt lõi của khảo sát. Tuy nhiên, trong phần này, chúng tôi xin phân biệt ngắn gọn 3 loại thuật toán.

· **Exact methods.** ILP models are generally solved to proven optimality (or proof of infeasibility) by variations of branch-&-bound algorithm (Land & Doig, 1960; Lodi, 2010). Essentially, algorithm is an iterative divide-&-conquer method that

1. solves LP relaxations (Sect. 2.3.1),
2. improves them through valid inequalities (or cutting planes), &
3. guarantees to find an optimal solution through implicit enumeration performed by branching, see Fig. 3: Variable selection in branch-&-bound integer programming algorithm as a MDP.

As anticipated in Sect. 2.3.1, quality of LP relaxation plays a fundamental role in effectiveness of above scheme. Thus, step 2 above is particularly important, especially at beginning of search. Above scheme is called *branch & cut*. If integer programming techniques do not directly model CO problem, then combinatorial versions of branch-&-bound framework are devised, i.e., featuring relaxations different from LP one, specifically associated with structure of CO problem at hand.

– **Phương pháp chính xác.** Các mô hình ILP thường được giải quyết để chứng minh tính tối ưu (hoặc chứng minh tính không khả thi) bằng các biến thể của thuật toán nhánh-&-bound (Land & Doig, 1960; Lodi, 2010). Về cơ bản, thuật toán là 1 phương pháp chia-&-trị lặp

1. giải các phép giãn LP (Phần 2.3.1),
2. cải thiện chúng thông qua các bất đẳng thức hợp lệ (hoặc các mặt phẳng cắt), &

3. đảm bảo tìm ra 1 giải pháp tối ưu thông qua phép liệt kê ngầm được thực hiện bằng cách rẽ nhánh, xem Hình 3: Lựa chọn biến trong thuật toán lập trình số nguyên nhánh-&-bound như 1 MDP.

Như đã dự đoán trong Phần 2.3.1, chất lượng giãn LP đóng vai trò cơ bản trong hiệu quả của lược đồ trên. Do đó, bước 2 ở trên đặc biệt quan trọng, đặc biệt là khi bắt đầu tìm kiếm. Lược đồ trên được gọi là *branch & cut*. Nếu các kỹ thuật lập trình số nguyên không trực tiếp mô hình hóa vấn đề CO, thì các phiên bản kết hợp của khuôn khổ liên kết nhánh sẽ được đưa ra, tức là, có các biện pháp nới lỏng khác với LP, liên quan cụ thể đến cấu trúc của vấn đề CO đang được xử lý.

Specifically designed as an exact method for constrained satisfaction & optimization problems, constraint programming (CP) (Rossi et al., 2006) is a general framework proposing algorithmic solutions also within divide-&-conquer scheme. It is a complete approach, i.e., possible to prove optimality of solutions found. Solving process consists of a complete enumeration of all possible variable assignments until best solution has been found. To cope with implied (exponentially) large search trees, one utilizes a mechanism called propagation, which reduces number of possibilities. Here, propagation of constraint c removes values from domain violated by c. This process is repeated at each domain change & for each constraint until no value exists any more. A CP solver's efficiency relies heavily on its propagators' quality. Example 4 introduced wellknown AllDifferent constraint. Its propagator (Régin, 1994) is based on maximum matching algorithms in a graph. Many other global constraints are available in literature, e.g. Element constraint, allowing indexation with variables, or Circuit constraint, enforcing a set of variables to create a valid circuit. At time of writing, global constraints catalog reports > 400 global constraints (Beldiceanu et al., 2005). CP search commonly proceeds in a depth-1st fashion, together with branch&-bound. For each feasible solution found, solver adds a constraint, ensuring: following solution has to be better than current one. Upon finding an infeasible solution, search backtracks to previous decision. With this procedure, & provided: whole search space has been explored, final solution is guaranteed to be optimal.

– Được thiết kế đặc biệt như 1 phương pháp chính xác cho các bài toán tối ưu hóa thỏa mãn ràng buộc, lập trình ràng buộc (CP) (Rossi & cộng sự, 2006) là 1 khuôn khổ chung đề xuất các giải pháp thuật toán cũng trong lược đồ chia-&-trị. Đây là 1 phương pháp tiếp cận hoàn chỉnh, tức là có thể chứng minh tính tối ưu của các giải pháp tìm được. Quá trình giải quyết bao gồm việc liệt kê đầy đủ tất cả các phép gán biến khả thi cho đến khi tìm thấy giải pháp tốt nhất. Để xử lý các cây tìm kiếm lớn hàm ý (theo cấp số nhân), người ta sử dụng 1 cơ chế gọi là lan truyền, giúp giảm số lượng khả năng. Ở đây, lan truyền ràng buộc c loại bỏ các giá trị khỏi miền bị vi phạm bởi c. Quá trình này được lặp lại tại mỗi lần thay đổi miền & cho mỗi ràng buộc cho đến khi không còn giá trị nào tồn tại nữa. Hiệu quả của bộ giải CP phụ thuộc rất nhiều vào chất lượng của các bộ lan truyền của nó. Ví dụ 4 đã giới thiệu ràng buộc AllDifferent nổi tiếng. Bộ lan truyền của nó (Régin, 1994) dựa trên các thuật toán khớp tối đa trong 1 đồ thị. Nhiều ràng buộc toàn cục khác có sẵn trong tài liệu, ví dụ: Ràng buộc phần tử, cho phép lập chỉ mục bằng biến, hoặc ràng buộc mạch, áp đặt 1 tập hợp các biến để tạo ra 1 mạch hợp lệ. Tại thời điểm viết bài, danh mục ràng buộc toàn cục báo cáo > 400 ràng buộc toàn cục (Beldiceanu & cộng sự, 2005). Tìm kiếm CP thường tiến hành theo chiều sâu, kết hợp với ràng buộc nhánh. Với mỗi giải pháp khả thi được tìm thấy, trình giải thêm 1 ràng buộc, đảm bảo: giải pháp tiếp theo phải tốt hơn giải pháp hiện tại. Khi tìm thấy 1 giải pháp không khả thi, tìm kiếm sẽ quay lại quyết định trước đó. Với quy trình này, với điều kiện: toàn bộ không gian tìm kiếm đã được khám phá, giải pháp cuối cùng được đảm bảo là tối ưu.

Finally, although initially designed for solving decision problem, SAT solvers can also be used for combinatorial optimization. 1 way to do that: specify objectives through soft constraints. Objective turns to satisfy as many soft constraints as possible in a solution, e.g. MaxSAT variant. Another option: add a repertoire of commonly used objective functions in solver & invoke specialized optimization module when it corresponds to objective function that must be optimized. Satisfiability module theories (SMT) solvers, a generalization of SAT, which can handle more complex formulas, generally support both options; see e.g. Z3 solver (de Moura & Bjørner, 2008).

– Cuối cùng, mặc dù ban đầu được thiết kế để giải quyết bài toán quyết định, các bộ giải SAT cũng có thể được sử dụng cho tối ưu hóa tổ hợp. 1 cách để làm điều đó: xác định mục tiêu thông qua các ràng buộc mềm. Mục tiêu xoay quanh việc thỏa mãn càng nhiều ràng buộc mềm càng tốt trong 1 giải pháp, ví dụ: biến thể MaxSAT. 1 lựa chọn khác: thêm 1 danh mục các hàm mục tiêu thường dùng vào bộ giải & gọi mô-đun tối ưu hóa chuyên biệt khi nó tương ứng với hàm mục tiêu cần được tối ưu hóa. Các bộ giải lý thuyết mô-đun thỏa mãn (SMT), 1 dạng tổng quát của SAT, có thể xử lý các công thức phức tạp hơn, thường hỗ trợ cả 2 tùy chọn; xem ví dụ: bộ giải Z3 (de Moura & Bjørner, 2008).

· **Local search & metaheuristics.** Local search (Potvin & Gendreau, 2018) is another algorithmic framework commonly used to solve general, large-scale CO problems. Local search only partially explores solution space in a perturbative fashion & is thus an incomplete approach that does not provide an optimality guarantee on solution it returns. In its simplest form, search starts from a candidate solution s & iteratively explores solution space by selecting a neighboring solution until no improvement occurs. Here, a solution's neighborhood is set of solutions obtained by modifying solution s. In practice, local search algorithms are improved through metaheuristics concepts (Glover & Kochenberger, 2006), leading to algorithms like simulated annealing (Van Laarhoven & Aarts, 1987; Delahaye et al., 2019), tabu search (Glover & Laguna, 1998; Laguna, 2018), genetic algorithms (Kramer, 2017), variable neighborhood search (Mladenović & Hansen, 1997; Hansen et al., 2019), all of which are designed to help escape local minima.

– **Tìm kiếm cục bộ & siêu thuật toán.** Tìm kiếm cục bộ (Potvin & Gendreau, 2018) là 1 khuôn khổ thuật toán khác thường được sử dụng để giải các bài toán CO tổng quát, quy mô lớn. Tìm kiếm cục bộ chỉ khám phá 1 phần không gian nghiệm theo cách nhiễu động & do đó là 1 phương pháp chưa hoàn chỉnh, không đảm bảo tính tối ưu cho nghiệm mà nó trả về. Ở dạng đơn giản nhất, tìm kiếm bắt đầu từ 1 nghiệm ứng viên s & lặp lại khám phá không gian nghiệm bằng cách chọn 1 nghiệm lân cận cho đến khi không còn cải thiện nào xảy ra. Ở đây, lân cận của 1 nghiệm là tập hợp các nghiệm thu được bằng cách sửa đổi nghiệm s. Trên thực tế, các thuật toán tìm kiếm cục bộ được cải thiện

thông qua các khái niệm siêu thuật toán (Glover & Kochenberger, 2006), dẫn đến các thuật toán như ủ mô phỏng (Van Laarhoven & Aarts, 1987; Delahaye & cộng sự, 2019), tìm kiếm tabu (Glover & Laguna, 1998; Laguna, 2018), thuật toán di truyền (Kramer, 2017), tìm kiếm lân cận biến đổi (Mladenović & Hansen, 1997; Hansen & cộng sự, 2019), tất cả đều được thiết kế để giúp thoát khỏi các cực tiểu cục bộ.

· **Approximation algorithms.** Class of approximation algorithms (Vazirani, 2010) is designed to produce, typically in polynomial t ime, feasible solutions for CO problems. Unlike local search & metaheuristics, value of those feasible solutions is guaranteed to be within a certain bound from optimal one. Notable examples of approximation algorithms are polynomial-time approximation schemes (PTAS) that provide a solution that is within a factor $1 + \epsilon$ (with $\epsilon > 0$ being an input for algorithm) of being optimal (e.g., Arora (1996) for TSP), or fully polynomial-time approximation schemes (FPTAS), where additional conditions on running time of algorithm are imposed (Ausiello et al., 2012).
– **Thuật toán xấp xỉ.** Lớp thuật toán xấp xỉ (Vazirani, 2010) được thiết kế để tạo ra, thường trong thời gian đa thức, các giải pháp khả thi cho các bài toán CO. Không giống như tìm kiếm cục bộ & metaheuristic, giá trị của các giải pháp khả thi này được đảm bảo nằm trong 1 giới hạn nhất định so với giá trị tối ưu. Các ví dụ đáng chú ý về thuật toán xấp xỉ là các lược đồ xấp xỉ thời gian đa thức (PTAS) cung cấp 1 giải pháp nằm trong hệ số $1 + \epsilon$ (với $\epsilon > 0$ là đầu vào của thuật toán) để đạt được tính tối ưu (ví dụ: Arora (1996) cho TSP), hoặc các lược đồ xấp xỉ thời gian đa thức đầy đủ (FPTAS), trong đó các điều kiện bổ sung về thời gian chạy của thuật toán được áp dụng (Ausiello & cộng sự, 2012).

◦ 2.4. ML. This sect gives a short & concise overview of ML. Cover 3 main branches of field, i.e., supervised learning, unsupervised learning, & reinforcement learning. For details, see Mohri et al. (2012); Shalev-Shwartz & Ben-David (2014). Moreover, introduce imitation learning highly relevant to CO.

– Phần này cung cấp 1 cái nhìn tổng quan ngắn gọn về Học máy. Bao gồm 3 nhánh chính của lĩnh vực này, tức là học có giám sát, học không giám sát & học tăng cường. Để biết thêm chi tiết, xem Mohri & cộng sự (2012); Shalev-Shwartz & Ben-David (2014). Ngoài ra, giới thiệu về học bắt chước, 1 phương pháp học tập có liên quan mật thiết đến Học máy.

∗ **Supervised learning.** +++

◦ 2.5. Graph Neural networks. +++

• 3. GNNs for Combinatorial Optimization: State of Art. Given that many practically relevant CO problems are NP-hard, helpful to characterize algorithms for solving them as prioritizing 1 of 2 goals. Primal goal of finding good feasible solutions, & dual goal of certifying optimality or proving infeasibility. In both cases, GNNs can serve as a tool for representing problem instances, states of an iterative algorithm, or both. Not uncommon to combine GNNs' variable or constraint representations with hand-crafted features, which would otherwise be challenging to extract automatically with GNN. Coupled with an appropriate ML paradigm (Sect. 2.4), GNNs have been shown to guide exact & heuristic algorithms towards finding good feasible solutions faster (Sect. 3.1). GNNs have also been used to guide certifying optimality or infeasibility more efficiently (Sect. 3.2). In this case, GNNs are usually integrated with an existing complete algorithm because an optimality certificate has in general, exponential size concerning problem description size, & not clear how to devise GNNs with such large outputs. Beyond using standard GNN models for CO, emerging paradigm of algorithmic reasoning provides new perspectives on designing & training GNNs that satisfy natural invariants & properties, possibly enabling improved generalization & interpretability.

– GNN cho Tối ưu hóa Tổ hợp: Tình trạng Nghệ thuật. Do nhiều bài toán CO có liên quan thực tế là NP-khó, nên việc mô tả các thuật toán để giải chúng theo thứ tự ưu tiên 1 trong 2 mục tiêu là rất hữu ích. Mục tiêu chính là tìm ra các giải pháp khả thi tốt, & mục tiêu kép là chứng nhận tính tối ưu hoặc chứng minh tính không khả thi. Trong cả 2 trường hợp, GNN có thể đóng vai trò là công cụ để biểu diễn các trường hợp bài toán, trạng thái của 1 thuật toán lặp, hoặc cả hai. Việc kết hợp các biểu diễn biến hoặc ràng buộc của GNN với các đặc trưng thủ công, nếu không sẽ rất khó để trích xuất tự động bằng GNN, là điều không hiếm gặp. Kết hợp với 1 mô hình ML phù hợp (Phần 2.4), GNN đã được chứng minh là hướng dẫn các thuật toán & heuristic chính xác để tìm ra các giải pháp khả thi tốt nhanh hơn (Phần 3.1). GNN cũng đã được sử dụng để hướng dẫn việc chứng nhận tính tối ưu hoặc tính không khả thi hiệu quả hơn (Phần 3.2). Trong trường hợp này, GNN thường được tích hợp với 1 thuật toán hoàn chỉnh hiện có vì chứng chỉ tối ưu nhìn chung có kích thước theo cấp số nhân liên quan đến kích thước mô tả vấn đề, & không rõ cách thiết kế GNN với đầu ra lớn như vậy. Ngoài việc sử dụng các mô hình GNN tiêu chuẩn cho CO, mô hình lập luận thuật toán mới nổi cung cấp những góc nhìn mới về thiết kế & huấn luyện GNN thỏa mãn các thuộc tính bất biến tự nhiên, có thể cho phép cải thiện khả năng khái quát hóa & diễn giải.

◦ 3.1. On Primal Side: Finding Feasible Solutions. Begin by discussing use of GNNs in improving solution-finding process in CO. Following practical scenarios motivate need to quickly obtain high-quality feasible solutions, even without optimality or approximation guarantees.

– Về Mặt Nguyên Thủy: Tìm kiếm Giải pháp Khả thi. Hãy bắt đầu bằng việc thảo luận về việc sử dụng GNN để cải thiện quy trình tìm kiếm giải pháp trong CO. Các tình huống thực tế sau đây thúc đẩy nhu cầu nhanh chóng có được các giải pháp khả thi chất lượng cao, ngay cả khi không có sự đảm bảo về tính tối ưu hoặc xấp xỉ.

1. **Optimality guarantees are often no needed.** A practitioner may only be interested in quality of a feasible solution in absolute terms rather than relative to optimal value of a problem instance. E.g., if objective value represents financial cost, it might only care about how much profit is gained from switching from 1 resolution method to another. In this scenario then, heuristics are all that are needed. Moreover, there are many CO problems which are both practically intractable with an exact solver, & for which no proxy in form of a good dual bound is easily computable as well. For these problems, a practitioner has no other guide to assess suitability of a resolution method than to compare absolute objective values as well. E.g., many vehicle routing problems admit strong MIP formulations with an exponential number of variables or

constraints, similar to TSP formulation in Example 2, see Toth & Vigo (2014). For such problems, simply computing linear relaxation (typically using column or constraint generation (Dror et al., 1994))is challenging, so a heuristic that consistently finds good solutions within a short user-defined time limit might be preferable.

– **Đảm bảo tính tối ưu thường không cần thiết.** Người thực hành có thể chỉ quan tâm đến chất lượng của 1 giải pháp khả thi theo nghĩa tuyệt đối thay vì giá trị tương đối của 1 trường hợp bài toán. Ví dụ: nếu giá trị khách quan biểu thị chi phí tài chính, họ có thể chỉ quan tâm đến việc lợi nhuận thu được từ việc chuyển đổi từ phương pháp giải quyết này sang phương pháp khác là bao nhiêu. Trong trường hợp này, chỉ cần sử dụng phương pháp tìm kiếm (heuristics). Hơn nữa, có nhiều bài toán CO vừa khó giải quyết trên thực tế với 1 trình giải chính xác, vừa không thể tính toán dễ dàng bằng bất kỳ phương pháp đại diện nào dưới dạng 1 ràng buộc kép tốt. Đối với những bài toán này, người thực hành không có hướng dẫn nào khác để đánh giá tính phù hợp của 1 phương pháp giải quyết ngoài việc so sánh các giá trị khách quan tuyệt đối. Ví dụ: nhiều bài toán định tuyến xe cộ cho phép các công thức MIP mạnh với số mũ các biến hoặc ràng buộc, tương tự như công thức TSP trong Ví dụ 2, xem Toth & Vigo (2014). Đối với những vấn đề như vậy, việc chỉ tính toán sự giãn nở tuyến tính (thường sử dụng cột hoặc thế hệ ràng buộc (Dror & cộng sự, 1994)) là 1 thách thức, do đó, 1 phương pháp tìm kiếm liên tục tìm ra các giải pháp tốt trong giới hạn thời gian ngắn do người dùng xác định có thể là lựa chọn tốt hơn.

2.2 Martin J. A. Schuetz, J. Kyle Brubaker, Helmut G. Katzgraber. Combinatorial Optimization with Physics-Inspired GNNs

- Abstract. Combinatorial optimization problems are pervasive across science & industry. Modern DL tools ar poised to solve these problems at unprecedented scales, but a unifying framework that incorporates insights from statistical physics is still outstanding. Demonstrate how GNNs can be used to solve combinatorial optimization problems. Our approach is broadly applicable to canonical NP-hard problems in form of quadratic unconstrained binary optimization problems, e.g. maximum cut, minimum vertex cover, maximum independent set, as well Ising spin glasses & higher-order generalizations thereof in form of polynomial unconstrained binary optimization problems. Apply a relaxation strategy to problem Hamiltonian to generate a differentiable loss function with which we train GNN & apply a simple projection to integer variables once unsupervised training process has completed. Showcase our approach with numerical results for canonical maximum cut & maximum independent set problems. Find: GNN optimizer performs on par or outperforms existing solvers, with ability to scale beyond state of art to problems with millions of variables.

  – Các bài toán tối ưu hóa tổ hợp đang lan rộng trong khoa học & công nghiệp. Các công cụ DL hiện đại sẵn sàng giải quyết những bài toán này ở quy mô chưa từng có, nhưng 1 khuôn khổ thống nhất kết hợp những hiểu biết từ vật lý thống kê vẫn còn rất mới mẻ. Trình bày cách sử dụng GNN để giải các bài toán tối ưu hóa tổ hợp. Phương pháp của chúng tôi có thể áp dụng rộng rãi cho các bài toán NP-khó chính tắc dưới dạng các bài toán tối ưu hóa nhị phân bậc 2 không ràng buộc, e.g.: đường cắt cực đại, độ phủ đỉnh cực tiểu, tập độc lập cực đại, cũng như kính spin Ising & các tổng quát hóa bậc cao hơn của chúng dưới dạng các bài toán tối ưu hóa nhị phân không ràng buộc đa thức. Áp dụng chiến lược thư giãn cho Hamiltonian bài toán để tạo ra 1 hàm mất mát khả vi mà chúng tôi dùng để huấn luyện GNN & áp dụng 1 phép chiếu đơn giản lên các biến nguyên sau khi quá trình huấn luyện không giám sát hoàn tất. Trình bày phương pháp của chúng tôi với các kết quả số cho các bài toán đường cắt cực đại chính tắc & tập độc lập cực đại. Tìm: Trình tối ưu hóa GNN hoạt động ngang bằng hoặc vượt trội hơn các trình giải hiện có, với khả năng mở rộng vượt xa công nghệ tiên tiến cho các bài toán với hàng triệu biến.

- 1. Introduction. Optimization is ubiquitous across science & industry. Specifically, field of combinatorial optimization – search for minimum of an objective function within a finite but often large set of candidate solutions – is 1 of most important areas in field of optimization, with practical (yet notorious challenging) applications found in virtually every industry, including both private & public sectors, as well as in areas e.g. transportation & logistics, telecommunications, & finance. While efficient specialized algorithms exist for specific use cases, most optimization problems remain intractable, especially in real-world applications where problems are more structured & thus require additional steps to make them amenable to traditional optimization techniques. Despite remarkable advances in both algorithms & computing power, significant yet generic improvements have remained elusive, generating an increased interest in new optimization approaches that are broadly applicable & radically different from traditional operations research tools.

  – Tối ưu hóa hiện diện khắp nơi trong khoa học & công nghiệp. Cụ thể, lĩnh vực tối ưu hóa tổ hợp – tìm kiếm giá trị nhỏ nhất của 1 hàm mục tiêu trong 1 tập hợp hữu hạn nhưng thường lớn các giải pháp ứng viên – là 1 trong những lĩnh vực quan trọng nhất trong lĩnh vực tối ưu hóa, với các ứng dụng thực tế (nhưng đầy thách thức) được tìm thấy trong hầu hết mọi ngành, bao gồm cả khu vực tư nhân & công cộng, cũng như trong các lĩnh vực như vận tải & hậu cần, viễn thông & tài chính. Mặc dù có các thuật toán chuyên biệt hiệu quả cho các trường hợp sử dụng cụ thể, hầu hết các bài toán tối ưu hóa vẫn còn khó giải, đặc biệt là trong các ứng dụng thực tế, nơi các bài toán có cấu trúc hơn & do đó yêu cầu các bước bổ sung để làm cho chúng phù hợp với các kỹ thuật tối ưu hóa truyền thống. Bất chấp những tiến bộ đáng kể về cả thuật toán & sức mạnh tính toán, những cải tiến đáng kể nhưng chung chung vẫn còn khó nắm bắt, tạo ra sự quan tâm ngày càng tăng đối với các phương pháp tối ưu hóa mới có thể áp dụng rộng rãi & khác biệt hoàn toàn so với các công cụ nghiên cứu hoạt động truyền thống.

  In broader physics community, advent of quantum annealing devices e.g. D-Wave Systems Inc. quantum annealers [6–9] has spawned a renewed interest in development of heuristic approaches to solve discrete optimization problems. On 1 hand, recent advances in quantum science & technology have inspired development of novel classical algorithms, sometimes dubbed nature-inspired or physics-inspired algorithms (e.g., simulated quantum annealing [10, 11] running on conventional CMOS

hardware) that have raised bar for emerging quantum annealing hardware [12–15]. On other hand, in parallel to these algorithmic developments, substantial progress has been made in recent years on development of programmable special-purpose devices based on alternative technologies, e.g. coherent Ising machine based on optical parametric oscillators [16, 17], digital MemComputing machines based on self-organizing logic gates [18, 19], & ASIC-based Fujitsu Digital Annealer [20–22]. Some of these approaches face severe scalability limitations. E.g., in coherent Ising machine there is a trade off between precision & number of variables & Fujitsu Digital Annealer – baked into an ASIC – can currently handle at most 8192 variables. Thus, of much interest to find new alternate approaches to tackle large-scale combinatorial optimization problems, going far beyond what is currently accessible with quantum & nature-inspired approaches alike.

– Trong cộng đồng vật lý rộng lớn hơn, sự ra đời của các thiết bị ủ lượng tử, ví dụ như máy ủ lượng tử của D-Wave Systems Inc. [6–9] đã làm nảy sinh mối quan tâm mới trong việc phát triển các phương pháp tiếp cận theo phương pháp heuristic để giải quyết các vấn đề tối ưu hóa rời rạc. 1 mặt, những tiến bộ gần đây trong khoa học lượng tử & công nghệ đã truyền cảm hứng cho sự phát triển của các thuật toán cổ điển mới, đôi khi được gọi là các thuật toán lấy cảm hứng từ thiên nhiên hoặc lấy cảm hứng từ vật lý (e.g.: mô phỏng ủ lượng tử [10, 11] chạy trên phần cứng CMOS thông thường) đã nâng cao tiêu chuẩn cho phần cứng ủ lượng tử mới nổi [12–15]. Mặt khác, song song với những phát triển về thuật toán này, những tiến bộ đáng kể đã được thực hiện trong những năm gần đây về phát triển các thiết bị chuyên dụng có thể lập trình dựa trên các công nghệ thay thế, ví dụ như máy Ising mạch lạc dựa trên bộ dao động tham số quang học [16, 17], máy MemComputing kỹ thuật số dựa trên cổng logic tự tổ chức [18, 19], & Máy ủ kỹ thuật số Fujitsu dựa trên ASIC [20–22]. 1 số phương pháp này gặp phải những hạn chế nghiêm trọng về khả năng mở rộng. E.g., trong máy Ising mạch lạc, có sự đánh đổi giữa độ chính xác & số lượng biến & Fujitsu Digital Annealer – được tích hợp trong ASIC – hiện chỉ có thể xử lý tối đa 8192 biến. Do đó, việc tìm kiếm các phương pháp thay thế mới để giải quyết các bài toán tối ưu hóa tổ hợp quy mô lớn, vượt xa những gì hiện có thể tiếp cận được bằng các phương pháp lượng tử & lấy cảm hứng từ tự nhiên, là điều rất đáng quan tâm.

In DL community, GNNs have been a burst in popularity over last few years [23–30]. In essence, GNNs are deep neural network architectures specifically designed for graph structure data, with ability to learn effective feature representations of nodes, edges, or even entire graphs. Prime examples of GNN applications include classification of users in social networks [31, 32], prediction of future interactions in recommender systems [33], & prediction of certain properties of molecular graphs [34, 35]. As a convenient & general framework to model a variety of real-world complex structural data, GNNs have successfully been applied to a broad set of problems, including recommender systems in social media & e-commerce [36, 37], detection of misinformation (fake news) in social media [38], & various domains of natural sciences including event classification in particle physics [39, 40], to name a few. While several specific implementations of GNNs exist [29, 41, 42], at their core typically GNNs iteratively update features of nodes of a graph by aggregating information from their neighbors (often referred to as *message passing* [43]) thereby iteratively making local updates to graph structure as training of network progresses. Because of their scalability & inherent graph-based design, GNNs present an alternate platform to build large-scale combinatorial heuristics.

– Trong cộng đồng DL, GNN đã trở nên phổ biến trong vài năm trở lại đây [23–30]. Về bản chất, GNN là kiến trúc mạng nơ-ron sâu được thiết kế riêng cho dữ liệu cấu trúc đồ thị, với khả năng học các biểu diễn đặc trưng hiệu quả của các nút, cạnh hoặc thậm chí toàn bộ đồ thị. Các ví dụ điển hình về ứng dụng GNN bao gồm phân loại người dùng trong mạng xã hội [31, 32], dự đoán các tương tác trong tương lai trong hệ thống đề xuất [33], & dự đoán 1 số thuộc tính nhất định của đồ thị phân tử [34, 35]. Là 1 khuôn khổ chung thuận tiện để mô hình hóa nhiều loại dữ liệu cấu trúc phức tạp trong thế giới thực, GNN đã được áp dụng thành công cho nhiều vấn đề, bao gồm hệ thống đề xuất trong phương tiện truyền thông xã hội & thương mại điện tử [36, 37], phát hiện thông tin sai lệch (tin giả) trên phương tiện truyền thông xã hội [38], & nhiều lĩnh vực khoa học tự nhiên khác nhau bao gồm phân loại sự kiện trong vật lý hạt [39, 40], v.v. Mặc dù có 1 số triển khai cụ thể của GNN [29, 41, 42], nhưng về cơ bản, GNN thường cập nhật lặp lại các đặc điểm của các nút trong đồ thị bằng cách tổng hợp thông tin từ các nút lân cận (thường được gọi là *message passing* [43]), do đó thực hiện lặp lại các cập nhật cục bộ cho cấu trúc đồ thị khi quá trình huấn luyện mạng tiến triển. Nhờ khả năng mở rộng & thiết kế dựa trên đồ thị vốn có, GNN cung cấp 1 nền tảng thay thế để xây dựng các thuật toán tìm kiếm tổ hợp quy mô lớn.

In this work, present a highly-scalable GNN-based solver to (approximately) solve combinatorial optimization problems with up to millions of variables. Approach is schematically depicted in Fig. 1: Schematic illustration of GNN approach for combinatorial optimization presented in this work. Following a recursive neighborhood aggregation scheme, GNN is iteratively trained against a custom loss function that encodes specific optimization problem, e.g., maximum cut. At training completion, project final values for soft node assignments at final GNN layer back to binary variables $x_i = 0, 1$, providing solution bit string $\mathbf{x} = (x_1, x_2, \ldots)$. & works as follows: 1st, identify Hamiltonian (cost function) H that encodes optimization in terms of binary decision variables $x_v \in \{0, 1\}$ & associate this variable with a vertex $v \in V$ for an undirected graph $G = (V, E)$ with vertex set $V = [n]$ & edge set $E = \{(i, j) : i, j \in V\}$ capturing interactions between decision variables. Then apply a relaxation strategy to problem Hamiltonian to generate a differentiable loss function with which we perform unsupervised training on node representations of GNN. GNN follows a standard recursive neighborhood aggregation scheme [43, 44], where each node $v \in [n]$ collects information (encoded as feature vectors) of its neighbors to compute its new feature vector $\mathbf{h}_v^k$ at layer $k = 0, 1, \ldots, K$. After k iterations of aggregation, a node is represented by its transformed feature vector $\mathbf{h}_v^k$, which captures structural information within node's k-hop neighborhood [28]. For binary classification tasks typically use convolutional aggregation steps, followed by application of a nonlinear softmax activation function to shrink down final embeddings $\mathbf{h}_v^K$ to 1D soft (probabilistic) node assignments $p_v = \mathbf{h}_v^K \in [0, 1]$. Finally, once unsupervised training process has completed, apply a projection heuristic to map these soft assignments p_v back to integer variables $x_v \in \{0, 1\}$ using, e.g., $x_v = \texttt{int}(p_v)$. Numerically showcase our approach with results for canonical NP-hard optimization problems e.g. maximum cut (MaxCut) & maximum independent set (MIS), showing: our GNN-based approach can perform on par or even better than existing well-established solvers, while being broadly applicable

to a large class of optimization problems. Further, scalability of our approach opens up possibility of studying unprecedented problem sizes with hundreds of millions of nodes when leveraging distributed training in a mini-batch fashion on a cluster of machines as demonstrated recently in [45].

– Trong bài báo này, chúng tôi trình bày 1 bộ giải dựa trên GNN có khả năng mở rộng cao để (xấp xỉ) giải các bài toán tối ưu tổ hợp với tối đa hàng triệu biến. Phương pháp này được mô tả sơ đồ trong Hình 1: Minh họa sơ đồ phương pháp GNN cho tối ưu hóa tổ hợp được trình bày trong bài báo này. Theo 1 sơ đồ tổng hợp lân cận đệ quy, GNN được huấn luyện lặp lại theo 1 hàm mất mát tùy chỉnh mã hóa bài toán tối ưu cụ thể, e.g.: cắt cực đại. Khi hoàn tất đào tạo, chiếu các giá trị cuối cùng cho các phép gán nút mềm tại lớp GNN cuối cùng trở lại các biến nhị phân $x_i = 0, 1$, cung cấp chuỗi bit giải pháp $\mathbf{x} = (x_1, x_2, \ldots)$. & hoạt động như sau: 1. Đầu tiên, xác định Hamiltonian (hàm chi phí) H mã hóa tối ưu hóa theo các biến quyết định nhị phân $x_v \in \{0, 1\}$ & liên kết biến này với 1 đỉnh $v \in V$ cho đồ thị vô hướng $G = (V, E)$ với tập đỉnh $V = [n]$ & tập cạnh $E = \{(i, j) : i, j \in V\}$ nắm bắt các tương tác giữa các biến quyết định. Sau đó, áp dụng 1 chiến lược thư giãn cho Hamiltonian vấn đề để tạo ra 1 hàm mất mát khả vi mà chúng ta thực hiện đào tạo không giám sát trên các biểu diễn nút của GNN. GNN tuân theo 1 sơ đồ tổng hợp lân cận đệ quy chuẩn [43, 44], trong đó mỗi nút $v \in [n]$ thu thập thông tin (được mã hóa dưới dạng các vectơ đặc trưng) của các nút lân cận để tính toán vectơ đặc trưng mới $\mathbf{h}_v^k$ của nó tại lớp $k = 0, 1, \ldots, K$. Sau k lần lặp tổng hợp, 1 nút được biểu diễn bằng vectơ đặc trưng đã biến đổi $\mathbf{h}_v^k$ của nó, vectơ này nắm bắt thông tin cấu trúc trong lân cận k-hop của nút [28]. Đối với các tác vụ phân loại nhị phân thường sử dụng các bước tổng hợp tích chập, sau đó áp dụng hàm kích hoạt softmax phi tuyến tính để thu nhỏ các nhúng cuối cùng $\mathbf{h}_v^K$ thành các phép gán nút mềm (xác suất) 1D $p_v = \mathbf{h}_v^K \in [0, 1]$. Cuối cùng, sau khi quá trình đào tạo không giám sát hoàn tất, hãy áp dụng phương pháp heuristic chiếu để ánh xạ các phép gán mềm p_v này trở lại các biến số nguyên $x_v \in \{0, 1\}$ bằng cách sử dụng, e.g.: $x_v = \texttt{int}(p_v)$. Trình bày phương pháp của chúng tôi bằng số với các kết quả cho các bài toán tối ưu hóa NP-khó chuẩn, ví dụ như cắt cực đại (MaxCut) & tập độc lập cực đại (MIS), cho thấy: phương pháp dựa trên GNN của chúng tôi có thể hoạt động ngang bằng hoặc thậm chí tốt hơn các bộ giải đã được thiết lập tốt hiện có, đồng thời có thể áp dụng rộng rãi cho 1 lớp lớn các bài toán tối ưu hóa. Hơn nữa, khả năng mở rộng của phương pháp của chúng tôi mở ra khả năng nghiên cứu các kích thước bài toán chưa từng có với hàng trăm triệu nút khi tận dụng đào tạo phân tán theo kiểu lô nhỏ trên 1 cụm máy như đã được chứng minh gần đây trong [45].

Structure: In Sect. 2, provide some context for our work, discussing recent developments at cross-section between ML & combinatorial optimization. Sect. 3 summarizes basic concepts underlying our approach, as well as information on class of problems that this approach can solve. Sect. 4 outlines implementation of proposed GNN-based optimizer, followed by numerical experiments in Sect. 4. In Sect. 4, discuss potential real-world applications in industry. In Sect. 7 draw conclusions & give an outlook on future directions of research.

– Cấu trúc: Trong Phần 2, hãy cung cấp 1 số bối cảnh cho công việc của chúng tôi, thảo luận về những phát triển gần đây tại giao điểm giữa ML & tối ưu hóa tổ hợp. Phần 3 tóm tắt các khái niệm cơ bản làm nền tảng cho phương pháp của chúng tôi, cũng như thông tin về các loại vấn đề mà phương pháp này có thể giải quyết. Phần 4 phác thảo việc triển khai bộ tối ưu hóa dựa trên GNN được đề xuất, tiếp theo là các thử nghiệm số trong Phần 4. Trong Phần 4, hãy thảo luận về các ứng dụng thực tế tiềm năng trong công nghiệp. Trong Phần 7, hãy rút ra kết luận & đưa ra triển vọng về các hướng nghiên cứu trong tương lai.

- 2. Related work. Briefly review relevant existing literature, with goal to provide a detailed context for our work. Broadly speaking, our work makes a physics-inspired contribution to emerging cross-fertilization between combinatorial optimization & ML, where development of novel DL architectures has sparked a renewed interest in heuristics for solving NP-hard combinatorial optimization problems using neural networks, as extensively reviewed in [46, 47]. Leaving alternative, non-graph-based approaches as presented e.g. in [48] aside, in following short survey we focus on graph-based optimization problems – where modern DL architectures e.g. sequence models, attention mechanisms, & GNNs provide a natural tool set [46] – & primarily distinguish between approaches based on supervised learning, reinforcement learning, or unsupervised learning. This categorization can be refined further w.r.t. typical size of a problem solved by a specific approach & scope of solver (special-purpose vs. general-purpose).

  – Tóm tắt lại các tài liệu hiện có có liên quan, với mục tiêu cung cấp bối cảnh chi tiết cho công việc của chúng tôi. Nhìn chung, công trình của chúng tôi đóng góp lấy cảm hứng từ vật lý vào sự giao thoa mới nổi giữa tối ưu hóa tổ hợp & Học máy (ML), trong đó sự phát triển của các kiến trúc Học máy mới đã khơi dậy mối quan tâm mới về phương pháp tìm kiếm để giải các bài toán tối ưu hóa tổ hợp NP-khó bằng mạng nơ-ron, như đã được xem xét kỹ lưỡng trong [46, 47]. Bỏ qua các phương pháp tiếp cận thay thế, không dựa trên đồ thị như đã trình bày, ví dụ như trong [48], trong khảo sát ngắn sau đây, chúng tôi tập trung vào các bài toán tối ưu hóa dựa trên đồ thị – trong đó các kiến trúc Học máy hiện đại, ví dụ như mô hình chuỗi, cơ chế chú ý, & Mạng nơ-ron nhân tạo (GNN) cung cấp 1 bộ công cụ tự nhiên [46] – & chủ yếu phân biệt giữa các phương pháp dựa trên học có giám sát, học tăng cường hoặc học không giám sát. Phân loại này có thể được tinh chỉnh hơn nữa theo quy mô điển hình của 1 bài toán được giải quyết bằng 1 phương pháp tiếp cận cụ thể & phạm vi của trình giải (mục đích chuyên biệt so với mục đích chung).

  Supervised Learning. Majority of neural network-based approaches to combinatorial optimization are based on supervised learning, with goal to approximate some (typically complex, nonlinear) mapping from an input representation of problem to target solution, based on minimization of some empirical, handcrafted loss function. Early work was based on pointer networks which leverage sequence-to-sequence models to produce permutations over inputs of variable size, as, e.g., relevant for canonical traveling salesman problem (TSP) [49]. Since then, numerous studies have fused GNNs with various heuristics & search procedures to solve specific combinatorial optimization problems, e.g. quadratic assignment [50], graph matching [51], graph coloring [52], & TSP [53, 54]. As pointed out in [55], however, viability & performance of supervised approaches

critically depends on existence of large, labeled training data sets with previously optimized hard problem instances, resulting in a problematic chicken-&-egg scenario, further amplified by fact: hard to efficiently sample unbiased & representative labeled instances of NP-hard problems [56].

– **Học có giám sát.** Phần lớn các phương pháp tiếp cận dựa trên mạng nơ-ron để tối ưu hóa tổ hợp đều dựa trên học có giám sát, với mục tiêu xấp xỉ 1 số ánh xạ (thường là phức tạp, phi tuyến tính) từ biểu diễn đầu vào của vấn đề đến giải pháp mục tiêu, dựa trên tối thiểu hóa 1 số hàm mất mát thủ công, theo kinh nghiệm. Các công trình ban đầu dựa trên mạng con trỏ tận dụng các mô hình chuỗi-sang-chuỗi để tạo ra các hoán vị trên các đầu vào có kích thước biến, ví dụ như có liên quan đến bài toán người bán hàng du lịch chính tắc (TSP) [49]. Kể từ đó, nhiều nghiên cứu đã hợp nhất GNN với nhiều phương pháp tìm kiếm & thủ tục tìm kiếm khác nhau để giải quyết các vấn đề tối ưu hóa tổ hợp cụ thể, ví dụ như phép gán bậc 2 [50], khớp đồ thị [51], tô màu đồ thị [52], & TSP [53, 54]. Tuy nhiên, như đã chỉ ra trong [55], tính khả thi & hiệu suất của các phương pháp có giám sát phụ thuộc rất nhiều vào sự tồn tại của các tập dữ liệu đào tạo có nhãn lớn với các trường hợp bài toán khó được tối ưu hóa trước đó, dẫn đến tình huống con gà & quả trứng có vấn đề, được khuếch đại thêm bởi thực tế: khó lấy mẫu hiệu quả các trường hợp có nhãn không thiên vị & đại diện của các bài toán khó NP [56].

Reinforcement Learning. Critical need for training labels can be circumvented with Reinforcement Learning (RL) techniques that aim to learn a policy with goal of maximizing some expected reward function. Specifically, optimization problems can typically be described with a native objective function that can then serve as a reward function in an RL approach [46]. Motivated by challenges associated with need for optimal target solutions, Bello et al. extended pointer network architecture [49] to an actor-critic RL framework to train an approximate TSP solver, using a RNN encoder scheme & expected tour length as a reward signal [57]. Using a general RL framework based on a graph attention network architecture [42], significant improvements in accuracy on 2D Euclidean TSP have subsequently been presented in [58], getting close to optimal results for problems up to 100 nodes. Moreover, TSP variants with hard constraints have been analyzed in [59], with help of a multi-level RL framework in which each layer of a hierarchy learns a different policy, & from which actions can then be sampled. Finally, while majority of RL-based approaches have focused on TSP or variants thereof, Dai et al. proposed a combination of RL & graph embedding to learn efficient greedy meta-heuristics to incrementally construct a solution, & showcased their approach with numerical results for Minimum Vertex Cover, MaxCut, & TSP as test problems, for graphs with up to $\sim$ 1000–1200 nodes [60].

– **Học tăng cường.** Nhu cầu quan trọng đối với nhãn đào tạo có thể được bỏ qua bằng các kỹ thuật Học tăng cường (RL) nhằm mục đích học 1 chính sách với mục tiêu tối đa hóa 1 số hàm thưởng mong đợi. Cụ thể, các vấn đề tối ưu hóa thường có thể được mô tả bằng 1 hàm mục tiêu gốc sau đó có thể đóng vai trò là hàm thưởng trong phương pháp RL [46]. Được thúc đẩy bởi những thách thức liên quan đến nhu cầu về các giải pháp mục tiêu tối ưu, Bello & cộng sự đã mở rộng kiến trúc mạng con trỏ [49] thành khuôn khổ RL diễn viên-nhà phê bình để đào tạo 1 bộ giải TSP gần đúng, sử dụng lược đồ mã hóa RNN & độ dài hành trình mong đợi làm tín hiệu thưởng [57]. Sử dụng khuôn khổ RL chung dựa trên kiến trúc mạng chú ý đồ thị [42], những cải tiến đáng kể về độ chính xác trên TSP Euclidean 2D sau đó đã được trình bày trong [58], đạt gần đến kết quả tối ưu cho các vấn đề lên đến 100 nút. Hơn nữa, các biến thể TSP với các ràng buộc cứng đã được phân tích trong [59], với sự trợ giúp của 1 khuôn khổ RL đa cấp, trong đó mỗi lớp của 1 hệ thống phân cấp học 1 chính sách khác nhau, & từ đó các hành động có thể được lấy mẫu. Cuối cùng, trong khi phần lớn các phương pháp tiếp cận dựa trên RL tập trung vào TSP hoặc các biến thể của nó, Dai & cộng sự đã đề xuất 1 sự kết hợp của RL & nhúng đồ thị để học các siêu heuristic tham lam hiệu quả nhằm xây dựng dần dần 1 giải pháp, & trình bày phương pháp của họ với các kết quả số cho Phủ đỉnh tối thiểu, Cắt tối đa, & TSP làm bài toán kiểm tra, cho các đồ thị có tối đa $\sim$ 1000–1200 nút [60].

Unsupervised Learning. Conceptually, our work is most similar to those that aim to train neural networks in an unsupervised, end-to-end fashion, without need for labeled training sets [55]. Specifically, Toenshoff et al. have recently used a recurrent GNN architecture – dubbed RUN-CSP – to solve optimization problems that can be framed as maximum constraint satisfaction problems [61]. For other types of problems, e.g. maximum independent set problem, model relies on empirically-selected hand-crafted loss functions. Using language of constraint satisfaction problems, where system size is expressed in terms of both number of variables & number of constraints, authors solve problem instances of Maximum 2-satisfiability, 3-colorability, MaxCut & Maximum Independent Set with up to 5000 nodes, showing: RUN-CSP can compete with traditional approaches like greedy heuristics or semi-definite programming. Finally, by either optimizing a smooth relaxation of cut objective or applying a policy gradient, Yao et al. trained a GNN to specifically solve MaxCut problem, albeit at relatively small system sizes with up to 500 nodes [62] & without any details on runtime.

– **Học không giám sát.** Về mặt khái niệm, công trình của chúng tôi tương tự nhất với những công trình hướng đến việc huấn luyện mạng nơ-ron theo cách không giám sát, từ đầu đến cuối, không cần tập huấn luyện có nhãn [55]. Cụ thể, Toenshoff & cộng sự gần đây đã sử dụng kiến trúc GNN hồi quy – được gọi là RUN-CSP – để giải quyết các bài toán tối ưu hóa có thể được đóng khung thành các bài toán thỏa mãn ràng buộc tối đa [61]. Đối với các loại bài toán khác, ví dụ như bài toán tập độc lập tối đa, mô hình dựa trên các hàm mất mát được tạo thủ công theo kinh nghiệm. Sử dụng ngôn ngữ của các bài toán thỏa mãn ràng buộc, trong đó kích thước hệ thống được biểu thị theo cả số biến & số ràng buộc, các tác giả giải quyết các trường hợp bài toán về khả năng thỏa mãn tối đa 2, khả năng tô màu 3, MaxCut & Tập độc lập tối đa với tối đa 5000 nút, cho thấy: RUN-CSP có thể cạnh tranh với các phương pháp tiếp cận truyền thống như thuật toán heuristic tham lam hoặc lập trình bán xác định. Cuối cùng, bằng cách tối ưu hóa việc nới lỏng trơn tru mục tiêu cắt hoặc áp dụng 1 gradient chính sách, Yao & cộng sự đã đào tạo 1 GNN để giải quyết cụ thể vấn đề MaxCut, mặc dù ở quy mô hệ thống tương đối nhỏ với tối đa 500 nút [62] & không có bất kỳ chi tiết nào về thời gian chạy.

Here, present a highly-scalable, physics-inspired framework that uses DL tools in form of GNNs to approximate solutions to hard

combinatorial optimization problems with up to millions of variables. Our GNN optimizer with up to millions of variables. Our GNN optimizer is based on a direct mathematical relation between prototypical Ising spin Hamiltonians [63], Quadratic Binary Unconstrained Optimization (QUBO) & Polynomial Binary Unconstrained Optimization (PUBO) formalism & differentiable loss function with which we train GNN, thereby providing 1 unifying framework for a broad class of combinatorial optimization problems, & opening up powerful toolbox of statistical physics to modern DL approaches. Fusing concepts from statistical physics with modern ML tooling, propose a simple, generic, & robust solver that does not rely on hand-crafted loss functions. Specifically, show: same GNN optimizer can solve different QUBO problems, without any need to change architecture or loss function, while scaling to problem instances orders of magnitude larger than what many traditional QUBO solvers can handle [6, 12, 64, 65].

– Ở đây, xin giới thiệu 1 khuôn khổ có khả năng mở rộng cao, lấy cảm hứng từ vật lý, sử dụng các công cụ DL dưới dạng GNN để xấp xỉ các giải pháp cho các bài toán tối ưu hóa tổ hợp khó với tối đa hàng triệu biến. Bộ tối ưu hóa GNN của chúng tôi với tối đa hàng triệu biến. Bộ tối ưu hóa GNN của chúng tôi dựa trên mối quan hệ toán học trực tiếp giữa các Hamiltonian spin Ising nguyên mẫu [63], công thức Tối ưu hóa nhị phân không ràng buộc bậc 2 (QUBO) & Tối ưu hóa nhị phân không ràng buộc đa thức (PUBO) & hàm mất mát khả vi mà chúng tôi huấn luyện GNN, do đó cung cấp 1 khuôn khổ thống nhất cho 1 lớp rộng các bài toán tối ưu hóa tổ hợp, & mở ra bộ công cụ mạnh mẽ của vật lý thống kê cho các phương pháp DL hiện đại. Kết hợp các khái niệm từ vật lý thống kê với công cụ ML hiện đại, đề xuất 1 bộ giải đơn giản, chung chung, & mạnh mẽ không dựa vào các hàm mất mát được tạo thủ công. Cụ thể, hãy hiển thị: cùng 1 trình tối ưu hóa GNN có thể giải quyết các vấn đề QUBO khác nhau mà không cần phải thay đổi kiến trúc hoặc hàm mất mát, đồng thời mở rộng quy mô lên các trường hợp vấn đề lớn hơn nhiều lần so với những gì nhiều trình giải QUBO truyền thống có thể xử lý [6, 12, 64, 65].

- 3. Preliminaries. To set up our notation & terminology start out with a brief review of both combinatorial optimization, & GNNs.

  Combinatorial Optimization. Field of combinatorial optimization is concerned with settings where a large number of yes/no decisions must be made & each set of decisions yields a corresponding objective function value, like a cost or profit value, to be optimized [1]. Canonical combinatorial optimization problems include, among others, maximum cut problem (MaxCut), maximum independent set problem (MIS), minimum vertex cover problem, maximum clique problem & set cover problem. In all cases exact solutions are not feasible for sufficiently-large systems due to exponential growth of solution space as number of variables n increases. Bespole (approximate) algorithms to solve these problems can typically be identified, at cost of limited scope & generalizability. Conversely, in recent years QUBO framework has resulted in a powerful approach that unifies a rich variety of these NP-hard combinatorial optimization problems [1–3, 66]. Cost function for a QUBO problem can be expressed in compact form with following Hamiltonina (1)

$$H_{\text{QUBO}} = \mathbf{x}^\top Q \mathbf{x} = \sum_{i,j} x_i Q_{ij} x_j,$$

  where $\mathbf{x} = (x_1, x_2, \ldots)$: a vector of binary decision variables & QUBO matrix Q is a square matrix of constant numbers that encodes actual problem to solve. W.l.o.g., Q-matrix can be assumed to be symmetric or in upper triangular form [1]. Have omitted any irrelevant constant terms, as well as any linear terms as these can always be absorbed into Q-matrix because $x_i^2 = x_i$ for binary variables $x_i \in \{0, 1\}$. Problem constraints, as relevant for many real-world optimization problems, can be accounted for with help of penalty terms entering objective function (rather than being explicitly imposed), as detailed in [1]. Significance of QUBO problems is further illustrated by close relation to famous Ising model, which is known to provide mathematical formulations for many NP-complete & NP-hard problems, including all of Karp's 21 NP-complete problems [66]. As opposed to QUBO problems, Ising problems are described in terms of binary spin variables $z_i \in \{\pm 1\}$, that can be mapped straightforwardly to their equivalent QUBO form, & vice versa, using $z_i = 2x_i - 1$. By def, both QUBO & Ising models are quadratic, but can be naturally generalized to higher order PUBO problems, as described by N-local Hamiltonian (2)

$$H_{\text{PUBO}} = \sum_{k=0}^{N} \sum_{\langle i_1, i_2, \ldots, i_k \rangle} Q_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \cdots x_{i_k},$$

  with real coefficients $Q_{i_1 i_2 \ldots i_k}$, for some $N \geq 3$, & $\langle i_1, i_2, \ldots, i_k \rangle$ indicating a group of k binary variables (or spins in Ising formulation). Terms containing a product of k variables, of form $Q_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \cdots x_{i_k}$, are commonly referred to as k-local interactions with $Q_{i_1 i_2 \ldots i_k}$ being coupling constant. As exemplify below for some canonical problems, graph (hypergraph) problems can be naturally framed as QUBO (PUBO) problems. To this end, given an undirected graph $G = (V, E)$, simply associate a binary variable x_i with every vertex $i \in V$, & then express (node classification) objective as a QUBO problem, where specific assignment $\mathbf{x}$ can be visualized as a specific 2-tone (e.g., light & dark) color of graph [With coloring of graph, refer to a specific node classification as given by assignment vector $\mathbf{x}$, taking e.g. $x_i = 0$ as red node coloring & $x_i = 1$ as blue node coloring. Do not refer to well-known vertex coloring problem which seeks to color vertices of a graph s.t. no 2 adjacent vertices are of same color.], see Fig. 1.

  – **Tối ưu hóa tổ hợp.** Lĩnh vực tối ưu hóa tổ hợp liên quan đến các thiết lập trong đó phải đưa ra 1 số lượng lớn các quyết định có/không & mỗi tập hợp các quyết định tạo ra 1 giá trị hàm mục tiêu tương ứng, như giá trị chi phí hoặc lợi nhuận, cần được tối ưu hóa [1]. Các vấn đề tối ưu hóa tổ hợp chính tắc bao gồm, trong số những vấn đề khác, bài toán cắt cực đại (MaxCut), bài toán tập độc lập cực đại (MIS), bài toán phủ đỉnh cực tiểu, bài toán clique cực đại & bài toán phủ tập hợp. Trong mọi trường hợp, các giải pháp chính xác đều không khả thi đối với các hệ thống đủ lớn do không gian nghiệm tăng

theo cấp số nhân khi số biến n tăng. Các thuật toán Bespole (xấp xỉ) để giải các vấn đề này thường có thể được xác định, với chi phí là phạm vi hạn chế & khả năng khái quát hóa. Ngược lại, trong những năm gần đây, khuôn khổ QUBO đã dẫn đến 1 phương pháp tiếp cận mạnh mẽ thống nhất nhiều loại bài toán tối ưu hóa tổ hợp NP-khó này [1–3, 66]. Hàm chi phí cho bài toán QUBO có thể được biểu diễn dưới dạng rút gọn với Hamiltonina (1) sau đây

$$H_{\text{QUBO}} = \mathbf{x}^\top Q \mathbf{x} = \sum_{i,j} x_i Q_{ij} x_j,$$

trong đó $\mathbf{x} = (x_1, x_2, \ldots)$: 1 vectơ các biến quyết định nhị phân & Ma trận QUBO Q là 1 ma trận vuông các hằng số mã hóa bài toán thực tế cần giải. Ví dụ: ma trận Q có thể được coi là đối xứng hoặc ở dạng tam giác trên [1]. Đã bỏ qua bất kỳ hằng số không liên quan nào, cũng như bất kỳ hằng số tuyến tính nào vì chúng luôn có thể được đưa vào ma trận Q vì $x_i^2 = x_i$ đối với các biến nhị phân $x_i \in \{0, 1\}$. Các ràng buộc của vấn đề, liên quan đến nhiều vấn đề tối ưu hóa trong thế giới thực, có thể được giải thích bằng sự trợ giúp của các điều khoản phạt nhập vào hàm mục tiêu (thay vì được áp đặt 1 cách rõ ràng), như được trình bày chi tiết trong [1]. Tầm quan trọng của các vấn đề QUBO được minh họa thêm bằng mối quan hệ chặt chẽ với mô hình Ising nổi tiếng, được biết là cung cấp các công thức toán học cho nhiều vấn đề NP-hoàn chỉnh & NP-khó, bao gồm tất cả 21 vấn đề NP-hoàn chỉnh của Karp [66]. Ngược lại với các vấn đề QUBO, các vấn đề Ising được mô tả dưới dạng các biến spin nhị phân $z_i \in \{\pm 1\}$, có thể được ánh xạ trực tiếp sang dạng QUBO tương đương của chúng, & ngược lại, bằng cách sử dụng $z_i = 2x_i - 1$. Theo định nghĩa, cả 2 mô hình QUBO & Ising đều là bậc hai, nhưng có thể được tổng quát hóa 1 cách tự nhiên cho các bài toán PUBO bậc cao hơn, như được mô tả bởi Hamiltonian cục bộ N (2)

$$H_{\text{PUBO}} = \sum_{k=0}^{N} \sum_{\langle i_1, i_2, \ldots, i_k \rangle} Q_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \cdots x_{i_k},$$

với hệ số thực $Q_{i_1 i_2 \ldots i_k}$, với 1 số $N \geq 3$, & $\langle i_1, i_2, \ldots, i_k \rangle$ biểu thị 1 nhóm k biến nhị phân (hoặc spin trong công thức Ising). Các số hạng chứa tích của k biến, có dạng $Q_{i_1 i_2 \ldots i_k} x_{i_1} x_{i_2} \cdots x_{i_k}$, thường được gọi là tương tác cục bộ k với $Q_{i_1 i_2 \ldots i_k}$ là hằng số liên kết. Như minh họa dưới đây cho 1 số bài toán chính tắc, các bài toán đồ thị (siêu đồ thị) có thể được đóng khung tự nhiên thành các bài toán QUBO (PUBO). Để đạt được mục đích này, cho 1 đồ thị vô hướng $G = (V, E)$, chỉ cần liên kết 1 biến nhị phân x_i với mọi đỉnh $i \in V$, & sau đó biểu thị mục tiêu (phân loại nút) dưới dạng bài toán QUBO, trong đó phép gán cụ thể $\mathbf{x}$ có thể được hình dung dưới dạng màu 2 tông màu cụ thể (e.g.: sáng & tối) của đồ thị [Với việc tô màu đồ thị, hãy tham chiếu đến phân loại nút cụ thể được chỉ định bởi vectơ gán $\mathbf{x}$, lấy ví dụ $x_i = 0$ là tô màu nút đỏ & $x_i = 1$ là tô màu nút xanh. Không tham chiếu đến bài toán tô màu đỉnh nổi tiếng tìm cách tô màu các đỉnh của đồ thị nếu không có 2 đỉnh liền kề nào có cùng màu.], xem Hình 1.

Graph Neural Networks. On a high level, GNNs are a family of neural networks capable of learning how to aggregate information in graphs for purpose of representation learning. Typically, a GNN layer is comprised of 3 functions [35]:

1. a message passing function that permits information exchange between nodes over edges,
2. an aggregation function that combines collection of received messages into a single, fixed-length representation,
3. a (typically nonlinear) update activation function that produces node-level representations given previous layer representation & aggregated information.

While a single-layer GNN encapsulates a node's features based on its immediate or 1-hop neighborhood, by stacking multiple layers, model can propagate each node's features through intermediate nodes, analogous to broadening receptive field in downstream layers of convolutional neural networks. Formally, at layer $k = 0$, each node $v \in V$ is represented by some initial representation $\mathbf{h}_v^0 \in \mathbb{R}^{d_0}$, usually derived from node's label or given input features of dimensionality d_0 [68]. Following a recursive neighborhood aggregation scheme, GNN then iteratively updates each node's representation, in general described by some parametric function f_θ^k, resulting in (3)

$$\mathbf{h}_v^k = f_\theta^k(\mathbf{h}_v^{k-1}, \{\mathbf{h}_u^{k-1} | u \in \mathcal{N}_v\}),$$

for layers $k \in [K]$, with $\mathcal{N}_v = \{u \in V | (u, v) \in E\}$ referring to local neighborhood of node v, i.e., set of nodes that share edges with node v. Total number of layers K is usually determined empirically as a hyperparameter, as are intermediate representation dimensionality d_k. Both can be optimized in an outer loop. While a growing number of possible implementations for GNN architectures [30] exists, here use a graph convolutional network (GCN) [29] for which (3) reads explicitly as

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{h}_u^{k-1}}{|\mathcal{N}(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right),$$

with $\mathbf{W}_k, \mathbf{B}_k$ being (shared) trainable weight matrices, denominator $|\mathcal{N}(v)|$ serving as normalization factor (with other choices available as well) & $\sigma(\cdot)$ being some (component-wise) nonlinear activation function e.g. sigmoid or ReLU. While GNNs can be used for various prediction tasks (including node classification, link prediction, community detection, network similarity, or graph classification), here focus on node classification, where usually last K-th layer's output is used to predict a label y_v for every node $v \in V$. To this end, feed (parameterized) final node embeddings $\mathbf{z}_v = \mathbf{h}_v^K(\theta)$ into a problem-specific loss function & run stochastic gradient descent to train weight parameters.

– **Mạng nơ-ron đồ thị.** Ở cấp độ cao, GNN là 1 họ mạng nơ-ron có khả năng học cách tổng hợp thông tin trong đồ thị nhằm mục đích học biểu diễn. Thông thường, 1 lớp GNN bao gồm 3 hàm [35]:

1. 1 hàm truyền thông điệp cho phép trao đổi thông tin giữa các nút qua các cạnh,
2. 1 hàm tổng hợp kết hợp tập hợp các thông điệp đã nhận thành 1 biểu diễn duy nhất có độ dài cố định,
3. 1 hàm kích hoạt cập nhật (thường là phi tuyến tính) tạo ra các biểu diễn cấp nút dựa trên biểu diễn lớp trước đó & thông tin tổng hợp.

Trong khi GNN 1 lớp đóng gói các đặc trưng của 1 nút dựa trên vùng lân cận trực tiếp hoặc 1-hop của nó, bằng cách xếp chồng nhiều lớp, mô hình có thể truyền các đặc trưng của từng nút qua các nút trung gian, tương tự như việc mở rộng trường tiếp nhận trong các lớp hạ lưu của mạng nơ-ron tích chập. Về mặt hình thức, tại lớp $k = 0$, mỗi nút $v \in V$ được biểu diễn bằng 1 số biểu diễn ban đầu $\mathbf{h}_v^0 \in \mathbb{R}^{d_0}$, thường được lấy từ nhãn của nút hoặc các tính năng đầu vào được cung cấp của chiều d_0 [68]. Theo sơ đồ tổng hợp lân cận đệ quy, GNN sau đó lặp lại việc cập nhật biểu diễn của từng nút, thường được mô tả bởi 1 hàm tham số f_θ^k, dẫn đến (3)

$$\mathbf{h}_v^k = f_\theta^k(\mathbf{h}_v^{k-1}, \{\mathbf{h}_u^{k-1} | u \in \mathcal{N}_v\}),$$

đối với các lớp $k \in [K]$, với $\mathcal{N}_v = \{u \in V | (u, v) \in E\}$ tham chiếu đến lân cận cục bộ của nút v, i.e., tập hợp các nút chia sẻ cạnh với nút v. Tổng số lớp K thường được xác định theo kinh nghiệm dưới dạng siêu tham số, cũng như chiều biểu diễn trung gian d_k. Cả 2 đều có thể được tối ưu hóa trong 1 vòng lặp ngoài. Mặc dù ngày càng có nhiều triển khai khả thi cho kiến trúc GNN [30], ở đây chúng ta sử dụng mạng tích chập đồ thị (GCN) [29] mà (3) được đọc rõ ràng là

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{h}_u^{k-1}}{|\mathcal{N}(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right),$$

với $\mathbf{W}_k, \mathbf{B}_k$ là các ma trận trọng số có thể huấn luyện (chia sẻ), mẫu số $|\mathcal{N}(v)|$ đóng vai trò là hệ số chuẩn hóa (cũng có các lựa chọn khác) & $\sigma(\cdot)$ là 1 hàm kích hoạt phi tuyến tính (theo từng thành phần), e.g.: sigmoid hoặc ReLU. Mặc dù GNN có thể được sử dụng cho nhiều tác vụ dự đoán khác nhau (bao gồm phân loại nút, dự đoán liên kết, phát hiện cộng đồng, độ tương đồng mạng hoặc phân loại đồ thị), nhưng ở đây tập trung vào phân loại nút, trong đó đầu ra của lớp K cuối cùng thường được sử dụng để dự đoán nhãn y_v cho mọi nút $v \in V$. Để đạt được mục đích này, hãy đưa các nhúng nút cuối cùng (đã tham số hóa) $\mathbf{z}_v = \mathbf{h}_v^K(\theta)$ vào 1 hàm mất mát cụ thể của bài toán & chạy thuật toán giảm dần gradient ngẫu nhiên để huấn luyện các tham số trọng số.

- 4. Combinatorial Optimization with GNNs. Now detail how to use GNNs to solve combinatorial optimization problems, as schematically outlined in Fig. 2: Flow chart illustrating end-to-end workflow for proposed physics-inspired GNN optimizer. (a) problem is specified by a graph G with associated adjacency matrix A, & a cost function as described (e.g.) by QUBO Hamiltonian H_{QUBO}. Within QUBO framework cost function is fully captured by QUBO matrix Q, as illustrated for both MaxCut & MIS for a sample (undirected) graph with 5 vertices & 6 edges. (b) Problem setup is complemented by a training strategy that specifies GNN Ansatz, a choice of hyperparameters & a specific ML optimizer. (c) GNN is iteratively trained against a custom loss function $\mathcal{L}_{\text{QUBO}}(\theta)$ that encodes a relaxed version of underlying optimization problem as specified by cost function H_{QUBO}. Typically, a GNN layer operates by aggregating information within local 1-hop neighborhood (as illustrated by $k = 1$ circle for top node with label 0). By stacking layers one can extend receptive field of each node, thereby allowing distant propagation of information (as illustrated by $k = 2$ circle for top node with label 0). (d)–(e) GNN generates soft node assignments which can be viewed as class probabilities. Using some projection scheme, then project soft node assignments back to (hard) binary variables $x_i = 0, 1$ (as indicated by binary black/white node coloring), providing final solution bit string $\mathbf{x}$. To this end, frame combinatorial optimization problems as unsupervised node classification tasks, without need for any labeled data. Because nodes do not carry any inherent features, in our setup node embeddings $\mathbf{h}_v^0$ are initialized randomly. Warm-starting training process with pre-training (transfer learning) will be left for future research. Class of Hamiltonians described above are not differentiable & cannot be used straightforwardly within GNN training process. Therefore, for a given problem Hamiltonian H & graph G, generate a differentiable loss function $\mathcal{L}(\theta)$, as required for standard back-propagation, by promoting binary decision variables $x_i \in \{0, 1\}$ to continuous (parametrized) probability parameters $p_i(\theta)$ with following (heuristic) relaxation approach (5)

$$x_i \to p_i(\theta) \in [0, 1].$$

Soft assignments p_i can be viewed as class probabilities. They are generated by our GNN Ansatz as final node embeddings $p_i = \mathbf{h}_i^K \in [0, 1]$ at layer K, after application of a nonlinear softmax activation function. Then, they are used as input for loss function $\mathcal{L}(\theta)$. In particular, for QUBO-type problems: (6)

$$H_{\text{QUBO}} \to \mathcal{L}_{\text{QUBO}}(\theta) = \sum_{i,j} p_i(\theta) Q_{ij} p_j(\theta),$$

which is differentiable w.r.t. parameters of GNN model θ, & similarly for PUBO problems on hypergraphs with higher-order terms of form $p_i p_j p_k$, etc. thereby establishing a straightforward, general connection between combinatorial optimization problems, Ising Hamiltonians & GNNs. For training with gradient descent, standard ML optimizers e.g. ADAM can be used. Once (unsupervised) training process has completed, apply projection heuristics to map these soft assignments p_i back to integer variables $x_i = 0, 1$, using e.g. simply $x_i = \texttt{int}(p_i)$. Application of other, more sophisticated projection schemes will be left for future research. Note: any projection heuristics can be applied throughout training after every epoch, thereby increasing pool of solution candidates, at no additional computational cost. With GNN guiding search through solution space, one can then book keep all solution candidates identified throughout training & simply pick best solution found.

– 4. Tối ưu hóa tổ hợp với GNN. Bây giờ, hãy trình bày chi tiết cách sử dụng GNN để giải các bài toán tối ưu hóa tổ hợp, như được phác thảo sơ đồ trong Hình 2: Sơ đồ luồng minh họa quy trình làm việc đầu cuối cho bộ tối ưu hóa GNN lấy cảm hứng từ vật lý được đề xuất. (a) bài toán được chỉ định bởi đồ thị G với ma trận kề A liên kết, & 1 hàm chi phí như được mô tả (ví dụ) bởi QUBO Hamiltonian H_{QUBO}. Trong khuôn khổ QUBO, hàm chi phí được nắm bắt đầy đủ bởi ma trận QUBO Q, như được minh họa cho cả MaxCut & MIS đối với 1 đồ thị mẫu (vô hướng) có 5 đỉnh & 6 cạnh. (b) Thiết lập bài toán được bổ sung bởi 1 chiến lược đào tạo chỉ định GNN Ansatz, 1 lựa chọn các siêu tham số & 1 bộ tối ưu hóa ML cụ thể. (c) GNN được huấn luyện lặp đi lặp lại theo hàm mất mát tùy chỉnh $\mathcal{L}_{\mathrm{QUBO}}(\theta)$ mã hóa phiên bản nới lỏng của bài toán tối ưu hóa cơ bản được chỉ định bởi hàm chi phí H_{QUBO}. Thông thường, 1 lớp GNN hoạt động bằng cách tổng hợp thông tin trong vùng lân cận 1 bước nhảy cục bộ (như minh họa bằng $k = 1$ vòng tròn cho nút trên cùng có nhãn 0). Bằng cách xếp chồng các lớp, người ta có thể mở rộng trường tiếp nhận của mỗi nút, do đó cho phép truyền thông tin đi xa (như minh họa bằng $k = 2$ vòng tròn cho nút trên cùng có nhãn 0). (d)–(e) GNN tạo ra các phép gán nút mềm có thể được xem như xác suất lớp. Sử dụng 1 số lược đồ chiếu, sau đó chiếu các phép gán nút mềm trở lại các biến nhị phân (cứng) $x_i = 0, 1$ (như được chỉ ra bằng cách tô màu nút đen/trắng nhị phân), cung cấp chuỗi bit giải pháp cuối cùng $\mathbf{x}$. Để đạt được mục đích này, hãy đóng khung các bài toán tối ưu hóa tổ hợp như các tác vụ phân loại nút không giám sát, không cần bất kỳ dữ liệu có nhãn nào. Do các nút không mang bất kỳ đặc điểm cố hữu nào, nên trong thiết lập của chúng tôi, nhúng nút $\mathbf{h}_v^0$ được khởi tạo ngẫu nhiên. Quá trình huấn luyện khởi động ấm với tiền huấn luyện (học chuyển giao) sẽ được dành cho nghiên cứu trong tương lai. Lớp Hamiltonian được mô tả ở trên không khả vi & không thể được sử dụng trực tiếp trong quá trình huấn luyện GNN. Do đó, đối với 1 bài toán Hamiltonian H & đồ thị G cho trước, hãy tạo 1 hàm mất mát khả vi $\mathcal{L}(\theta)$, như yêu cầu cho lan truyền ngược chuẩn, bằng cách đưa các biến quyết định nhị phân $x_i \in \{0, 1\}$ vào các tham số xác suất liên tục (tham số hóa) $p_i(\theta)$ với phương pháp nới lỏng (heuristic) sau (5)

$$x_i \to p_i(\theta) \in [0, 1].$$

Các phép gán mềm p_i có thể được xem như các xác suất lớp. Chúng được tạo ra bởi GNN Ansatz của chúng tôi dưới dạng nhúng nút cuối cùng $p_i = \mathbf{h}_i^K \in [0, 1]$ tại lớp K, sau khi áp dụng hàm kích hoạt softmax phi tuyến tính. Sau đó, chúng được sử dụng làm đầu vào cho hàm mất mát $\mathcal{L}(\theta)$. Cụ thể, đối với các bài toán kiểu QUBO: (6)

$$H_{\mathrm{QUBO}} \to \mathcal{L}_{\mathrm{QUBO}}(\theta) = \sum_{i,j} p_i(\theta) Q_{ij} p_j(\theta),$$

có thể phân biệt được với các tham số của mô hình GNN θ, & tương tự đối với các bài toán PUBO trên siêu đồ thị với các số hạng bậc cao có dạng $p_i p_j p_k$, v.v., qua đó thiết lập 1 kết nối tổng quát, đơn giản giữa các bài toán tối ưu hóa tổ hợp, Ising Hamiltonian & GNN. Đối với huấn luyện với phương pháp giảm dần độ dốc, có thể sử dụng các bộ tối ưu hóa ML tiêu chuẩn, ví dụ như ADAM. Sau khi quá trình huấn luyện (không giám sát) hoàn tất, hãy áp dụng các phương pháp suy luận chiếu để ánh xạ các phép gán mềm p_i này trở lại các biến nguyên $x_i = 0, 1$, e.g., sử dụng $x_i = \mathtt{int}(p_i)$. Việc áp dụng các lược đồ suy luận chiếu khác, phức tạp hơn sẽ được dành cho nghiên cứu trong tương lai. Lưu ý: bất kỳ phương pháp suy luận chiếu nào cũng có thể được áp dụng trong suốt quá trình huấn luyện sau mỗi kỷ nguyên, do đó tăng số lượng ứng viên giải pháp mà không tốn thêm chi phí tính toán. Với GNN hướng dẫn tìm kiếm trong không gian giải pháp, người ta có thể lưu trữ tất cả các ứng viên giải pháp đã xác định trong suốt quá trình huấn luyện & chỉ cần chọn giải pháp tốt nhất được tìm thấy.

Our general GNN approach features several hyperparameters, including number of layers K, dimensionality of embedding vectors $\mathbf{h}_i^k$, & learning rate β, with details depending on specific architecture & optimizer used. These can be fine-tuned & optimized in an outer loop, using, e.g., standard techniques e.g. grid search or more advanced Bayesian optimization methods.

– Phương pháp GNN tổng quát của chúng tôi bao gồm 1 số siêu tham số, bao gồm số lớp K, số chiều của các vectơ nhúng $\mathbf{h}_i^k$, & tốc độ học β, với các chi tiết tùy thuộc vào kiến trúc & trình tối ưu hóa cụ thể được sử dụng. Những thông số này có thể được tinh chỉnh & tối ưu hóa trong 1 vòng lặp ngoài, sử dụng, e.g., các kỹ thuật tiêu chuẩn như tìm kiếm lưới hoặc các phương pháp tối ưu hóa Bayesian tiên tiến hơn.

Our GNN-based approach can be readily implemented with open-source libraries e.g. PyTorch Geometric [69] or Deep Graph Library [70]. Core of corresponding code is displayed in supplemental material for a GCN with 2 layers & a loss function for any QUBO problem. For illustration, an example solution to archetypal MaxCut problem (as implemented with this Ansatz) for a 3-regular graph with $n = 100$ vertices is shown in Fig. 3: Example solution to MaxCut for a random 3-regular graph with $n = 100$ nodes. After training completion, GNN provides a binary bit string $\mathbf{x}$ that assigns 1 of 2 possible colors (e.g., black or white) to each vertex. An edge is said to be cut when it connects 2 vertices of different colors. For a given graph, optimization problem is to assign colors in a way that as many edges as possible can be cut at same time (corresponding to antiferromagnetic ground-state of system). Here, cut size achieved with our GNN method amounts to 132.

– Cách tiếp cận dựa trên GNN của chúng tôi có thể dễ dàng được triển khai bằng các thư viện nguồn mở, e.g.: PyTorch Geometric [69] hoặc Thư viện Deep Graph [70]. Phần cốt lõi của mã tương ứng được hiển thị trong tài liệu bổ sung cho GCN có 2 lớp & hàm mất mát cho bất kỳ bài toán QUBO nào. Để minh họa, 1 giải pháp ví dụ cho bài toán MaxCut nguyên mẫu (được triển khai với Ansatz này) cho đồ thị 3-chính quy với $n = 100$ đỉnh được hiển thị trong Hình 3: Giải pháp ví dụ cho MaxCut cho đồ thị 3-chính quy ngẫu nhiên với $n = 100$ nút. Sau khi hoàn tất quá trình huấn luyện, GNN cung cấp 1 chuỗi bit nhị phân $\mathbf{x}$ gán 1 trong 2 màu có thể (e.g.: đen hoặc trắng) cho mỗi đỉnh. 1 cạnh được gọi là bị cắt khi nó kết nối 2 đỉnh có màu khác nhau. Đối với 1 đồ thị cho trước, vấn đề tối ưu hóa là gán màu sao cho có thể cắt được nhiều cạnh nhất có thể cùng 1 lúc (tương ứng với trạng thái cơ bản phản sắt từ của hệ thống). Ở đây, kích thước cắt đạt được bằng phương pháp GNN của chúng tôi là 132.

- 5. Numerical experiments. Perform numerical experiments using MaxCut & MIS benchmark problems. Before providing details on these numerical experiments, 1st describe our GNN model architecture as it is consistent across d-regular MaxCut & MIS problem instances described below. Certainly possible: better solutions can be found by fine-tuning hyper-parameters for every given problem instance. However, 1 of our goal: design a robust & scalable solver that is able to solve a large sample of instances efficiently without need of hand-tuning parameters on an instance-by-instance base.

  – Tiến hành các thí nghiệm số sử dụng các bài toán chuẩn MaxCut & MIS. Trước khi cung cấp chi tiết về các thí nghiệm số này, trước tiên hãy mô tả kiến trúc mô hình GNN của chúng tôi vì nó nhất quán trên các trường hợp bài toán MaxCut & MIS d-chính quy được mô tả bên dưới. Hoàn toàn có thể: các giải pháp tốt hơn có thể được tìm thấy bằng cách tinh chỉnh các siêu tham số cho mỗi trường hợp bài toán nhất định. Tuy nhiên, 1 trong những mục tiêu của chúng tôi là thiết kế 1 bộ giải mạnh mẽ & có khả năng mở rộng, có thể giải quyết 1 lượng lớn các trường hợp 1 cách hiệu quả mà không cần phải điều chỉnh thủ công các tham số trên cơ sở từng trường hợp.

  GNN Architecture. Use a simple 2-layer GCN architecture based on PyTorch GraphConv units. 1st convolutional layer is fed node embeddings of dimension d_0 & outputs a representation of size d_1. Next, apply a component-wise, nonlinear ReLU transformation. 2nd convolutional layer is then fed this intermediate representation & outputs output layer of size d_2, which is then fed through component-wise sigmoid transformation to provide a soft probability $p_i \in [0,1]$ for every node $i \in V$. Find: following simple heuristic for determining hyper-parameters d_0, d_1 works well: if number of nodes is large $n \geq 10^5$, then set $d_0 = \texttt{int}(\sqrt{n})$, else set $d_0 = \texttt{int}(\sqrt[3]{n})$, & take $d_1 = \texttt{int}(\frac{d_0}{2})$. Because solve for binary classification tasks, set final output dimension as $d_2 = 1$. However, for multi-color problems this could be extended to $C > 2$ classes by passing output layer through a softmax transformation (instead of a sigmoid) & taking argmax. Note: as graph size scales beyond $\sim 10^5$ nodes, memory becomes a concern, & so further reduce representations to allow GNN to be trained on a single GPU. Distributed training leveraging a whole cluster of machines will be discussed in Sect. 7. With GNN's output depending on random initialization of hidden feature vectors there is a risk of becoming stuck in a local optimum where GNN stops learning. To counter this issue, one can take multiple shots (i.e., run GNN training multiple times for different random seeds & choose best solution), thereby boosting performance at cost of extended runtime. In our numerical experiments, limited number of shots per instance to 5, only re-running training when an obviously sub-optimal solution was detected. Finally, set learning rate to $\beta = 10^{-4}$ & allow model to train for up to $\sim 10^5$ epochs, with a simple early stopping rule set to an absolute tolerance of 10^{-4} & a patience of 10^3.

  – **Kiến trúc GNN.** Sử dụng kiến trúc GCN 2 lớp đơn giản dựa trên các đơn vị PyTorch GraphConv. Lớp tích chập thứ nhất được nạp các nhúng nút có chiều d_0 & xuất ra 1 biểu diễn có kích thước d_1. Tiếp theo, áp dụng phép biến đổi ReLU phi tuyến tính từng thành phần. Lớp tích chập thứ 2 sau đó được nạp biểu diễn trung gian này & xuất ra lớp đầu ra có kích thước d_2, sau đó được đưa qua phép biến đổi sigmoid từng thành phần để cung cấp xác suất mềm $p_i \in [0,1]$ cho mọi nút $i \in V$. Tìm: phương pháp tìm kiếm đơn giản sau đây để xác định siêu tham số d_0, d_1 hoạt động tốt: nếu số nút lớn $n \geq 10^5$, thì đặt $d_0 = \texttt{int}(\sqrt{n})$, nếu không thì đặt $d_0 = \texttt{int}(\sqrt[3]{n})$, & lấy $d_1 = \texttt{int}(\frac{d_0}{2})$. Vì giải quyết cho các nhiệm vụ phân loại nhị phân, hãy đặt chiều đầu ra cuối cùng là $d_2 = 1$. Tuy nhiên, đối với các bài toán nhiều màu, điều này có thể được mở rộng thành $C > 2$ lớp bằng cách truyền lớp đầu ra qua phép biến đổi softmax (thay vì sigmoid) & lấy argmax. Lưu ý: khi kích thước đồ thị mở rộng vượt quá $\sim 10^5$ nút, bộ nhớ trở thành 1 mối quan tâm, & do đó giảm thêm các biểu diễn để cho phép GNN được huấn luyện trên 1 GPU duy nhất. Đào tạo phân tán tận dụng toàn bộ cụm máy sẽ được thảo luận trong Phần 7. Với đầu ra của GNN phụ thuộc vào việc khởi tạo ngẫu nhiên các vectơ đặc trưng ẩn, có nguy cơ bị kẹt trong 1 tối ưu cục bộ, nơi GNN dừng học. Để khắc phục vấn đề này, người ta có thể thực hiện nhiều lần (i.e., chạy đào tạo GNN nhiều lần cho các hạt giống ngẫu nhiên khác nhau & chọn giải pháp tốt nhất), do đó tăng hiệu suất với chi phí là thời gian chạy kéo dài. Trong các thử nghiệm số của chúng tôi, giới hạn số lần chụp cho mỗi trường hợp là 5, chỉ chạy lại đào tạo khi phát hiện ra 1 giải pháp rõ ràng là không tối ưu. Cuối cùng, đặt tốc độ học thành $\beta = 10^{-4}$ & cho phép mô hình đào tạo trong tối đa $\sim 10^5$ kỷ nguyên, với 1 quy tắc dừng sớm đơn giản được đặt thành dung sai tuyệt đối là 10^{-4} & độ kiên nhẫn là 10^3.

  Maximum Cut. MaxCut is an NP-hard combinatorial optimization problem with practical applications in machine scheduling [71], image recognition [72], & electronic circuit layout design [73]. In current era of noisy intermediate-scale quantum devices, with advent of novel hybrid quantum-classical algorithms e.g. Quantum Approximate Optimization Algorithm (QAOA) [74], MaxCut problem has recently attracted considerable attention as a potential use case of pre-error-corrected quantum devices, see [75–80]. MaxCut is a graph partitioning problem defined as follows: given a graph with vertex set V & edge set E, seek a partition of V into 2 subsets with maximum cut, where a cut refers to edges connecting 2 nodes from different vertex sets. Intuitively, i.e., score a point whenever an edge connects 2 nodes of different colors. To formulate MaxCut mathematically, introduce binary variables satisfying $x_i = 1$ if vertex i is in 1 set & $x_i = 0$ if it is in the other set. It is then easy to verify: quantity $x_i + x_j - 2x_i x_j = 1$ if edge (i,j) has been cut, & 0 otherwise. WIth help of adjacency matrix A_{ij} with $A_{ij} = 0$ if edge (i,j) does not exist & $A_{ij} > 0$ if a (possibly weighted) edge connects node i with j, MaxCut problem is described by following quadratic Hamiltonian: (7)

$$H_{\text{MaxCut}} = \sum_{i<j} A_{ij}(2x_i x_j - x_i - x_j)$$

  that falls into broader class of QUBO problems described by (1); provide explicit Q-matrix for a sample MaxCut problem in Fig. 2. Up to an irrelevant constant, MaxCut problem can equivalently by described by compact Ising Hamiltonian $H_{\text{MaxCut}} = \sum_{i<j} J_{ij} z_i z_j$ with $J_{ij} = \frac{A_{ij}}{2}$, favoring antiferromagnetic ordering of spins for $J_{ij} > 0$, as expected intuitively based on problem def. As our figure of merit, denote largest cut found as $\text{cut}^\star = -H_{\text{MaxCut}}(\mathbf{x}^\star)$ with $\mathbf{x}^\star$ referring to corresponding bit string.

  – **Cắt cực đại.** MaxCut là 1 bài toán tối ưu hóa tổ hợp NP-khó với các ứng dụng thực tế trong lập lịch máy [71], nhận dạng hình ảnh [72], & thiết kế bố trí mạch điện tử [73]. Trong thời đại hiện tại của các thiết bị lượng tử quy mô trung gian có

nhiễu, với sự ra đời của các thuật toán lượng tử-cổ điển lai mới ví dụ như Thuật toán tối ưu hóa xấp xỉ lượng tử (QAOA) [74], bài toán MaxCut gần đây đã thu hút được sự chú ý đáng kể như 1 trường hợp sử dụng tiềm năng của các thiết bị lượng tử được hiệu chỉnh trước lỗi, xem [75–80]. MaxCut là 1 bài toán phân vùng đồ thị được định nghĩa như sau: cho 1 đồ thị với tập đỉnh V & tập cạnh E, hãy tìm 1 phân vùng V thành 2 tập con có cắt cực đại, trong đó 1 cắt đề cập đến các cạnh kết nối 2 nút từ các tập đỉnh khác nhau. Theo trực giác, i.e., ghi 1 điểm bất cứ khi nào 1 cạnh kết nối 2 nút có màu khác nhau. Để xây dựng MaxCut về mặt toán học, hãy đưa vào các biến nhị phân thỏa mãn $x_i = 1$ nếu đỉnh i thuộc tập hợp 1 & $x_i = 0$ nếu nó thuộc tập hợp còn lại. Sau đó, dễ dàng kiểm tra: đại lượng $x_i + x_j - 2x_ix_j = 1$ nếu cạnh (i, j) đã bị cắt, & 0 nếu ngược lại. Với sự trợ giúp của ma trận kề A_{ij} với $A_{ij} = 0$ nếu cạnh (i, j) không tồn tại & $A_{ij} > 0$ nếu 1 cạnh (có thể có trọng số) kết nối nút i với j, bài toán MaxCut được mô tả bằng Hamiltonian bậc 2 sau: (7)

$$H_{\text{MaxCut}} = \sum_{i<j} A_{ij}(2x_ix_j - x_i - x_j)$$

thuộc lớp rộng hơn của các bài toán QUBO được mô tả bởi (1); cung cấp ma trận Q rõ ràng cho 1 bài toán MaxCut mẫu trong Hình 2. Với 1 hằng số không liên quan, bài toán MaxCut có thể được mô tả tương đương bằng Ising Hamiltonian compact $H_{\text{MaxCut}} = \sum_{i<j} J_{ij}z_iz_j$ với $J_{ij} = \frac{A_{ij}}{2}$, ưu tiên sắp xếp spin phản sắt từ cho $J_{ij} > 0$, như dự kiến 1 cách trực quan dựa trên định nghĩa của bài toán. Theo công trạng của chúng ta, hãy ký hiệu vết cắt lớn nhất tìm được là $\text{cut}^\star = -H_{\text{MaxCut}}(\mathbf{x}^\star)$ với $\mathbf{x}^\star$ tham chiếu đến chuỗi bit tương ứng.

Complexity of MaxCut depends on regularity & connectivity of underlying graph. Following an existing trend in community [76] , 1st consider MaxCut problem on random (unweighted) d-regular graphs, where every vertex is connected to exactly d other vertices. Perform benchmarks as follows. For graphs with up to a few hundred nodes, compare our GNN-based solver to (approximate) polynomial-time Goemans–Williamson (GW) algorithm [81], which provides current record for an approximate answer within some fixed multiplicative factor of optimum (referred to as approximation ratio α), using semidefinite programming & randomized rounding. Specifically, GW algorithm achieves a guaranteed approximation ratio of $\alpha \sim 0.878$ for generic graphs. This lower bound can be raised for specific graphs e.g. unweighted 3-regular graphs where $\alpha \sim 0.9326$ [82]. Our implementation of GW algorithm is based on open-source CVXOPT solver, with CVXPY as modeling interface. For very large graphs with up to a million nodes, numerical benchmarks are not available, but can compare our best solution $\text{cut}^\star$ to an analytical result derived in [83], where shown: with high probability (in limit $n \to \infty$) size of maximum cut for random d-regular graphs with n nodes is given by $\text{cut}^\star = (\frac{d}{4} + P_*\sqrt{\frac{d}{4}} + O(\sqrt{d}))n + O(n)$. Here $P_* \approx 0.7632$ refers to an universal constant related to ground-state energy of Sherrington–Kirkpatrick model [84, 85] that can be expressed analytically via Parisi's formula [83]. Thus take $\text{cut}_{\text{ub}} = \left(\frac{d}{4} + P_*\sqrt{\frac{d}{4}}\right) n$ as an upper-bound estimate for maximum cut size in large-n limit. Complement this upper bound with a lower bound as achieved by a simple, randomized 0.5-approximation algorithm that (on average) cuts half of edges, yielding a cut size of $\text{cut}_{\text{rnd}} \approx \frac{d}{4}n$ for a d-regular graph with $|E| = \frac{d}{2}n$. Our results for achieved cut size as a function of number of vertice n are shown in Fig. 4: Numerical results for MaxCut. Left panel; Average cut size for d-regular graphs with $d = 3$ & $d = 5$ as a function of number of vertices n, bootstrap-averaged over 20 random graph instances, for both GNN-based method & Goemans–Williamson (GW) algorithm. On each graph instance, GNN solver is allowed up to 5 shots, & GW algorithm takes 100 shots. Solid lines for $n \geq 10^3$ represent theoretical upper bounds. Inset: Estimated relative approximation ratio defined as $\frac{\text{cut}^\star}{\text{cut}_{\text{ub}}}$ shows: our approach consistently achieves high-quality solutions. Right panel: Algorithm runtime in secs for both GNN solver & GW algorithm. Errors bars refer to twice bootstrapped standard deviations, sampled across 20 random graph instances for every data point. All results are bootstrapped estimates of mean, with error bars denoting twice bootstrapped standard deviations, sampled across 20 random d-regular graphs for every data point. For graphs with up to a few hundred nodes, find: a simple 2-layer GCN architecture can perform on par with GW algorithm, while showing a runtime advantage compared to GW starting at around $n \approx 100$ nodes. For large graphs with $n \approx 10^4$ to 10^6 nodes, find: our approach consistently achieves high-quality solutions with $\text{cut}^\star \gtrsim 0.9\text{cut}_{\text{ub}}$ for both $d = 3, d = 5$, resp. (i.e., much better than any naive randomized algorithm). As expected for d-regular graphs, find $\text{cut}^\star$ to scale linearly with number of nodes n, i.e., $\text{cut}^\star \approx \gamma_d n$ with $\gamma_3 \approx 1.28$ & $\gamma_5 \approx 1.93$ for $d = 3, d = 5$, resp. Moreover, utilizing modern GPU hardware, observe a favorable runtime scaling at intermediate & large system sizes that allows us to solve instances with $n = 10^6$ nodes in approximately 10 minutes (which includes both GNN model training & post-processing steps). Specifically, as shown in Fig. 4, observe an approximately linear scaling of total runtime with $\sim n$, for large d-regular graphs with $10^5 \leq n \leq 10^6$; contrasted with observed GW algorithm scaling as $\sim n^{3.5}$ for problem sizes in range $n \lesssim 250$, thereby showing (expected) time complexity $\tilde{O}(n^{3.5})$ of interior-point method (as commonly used for solving semidefinite program underlying GW algorithm) that dominates GW algorithm runtime [86, 87].

– Độ phức tạp của MaxCut phụ thuộc vào tính chính quy & kết nối của đồ thị cơ sở. Theo xu hướng hiện có trong cộng đồng [76], trước tiên hãy xem xét bài toán MaxCut trên đồ thị d-chính quy ngẫu nhiên (không có trọng số), trong đó mỗi đỉnh được kết nối với đúng d đỉnh khác. Thực hiện các phép so sánh như sau. Đối với đồ thị có tối đa vài trăm nút, hãy so sánh bộ giải dựa trên GNN của chúng tôi với thuật toán Goemans–Williamson (GW) thời gian đa thức (xấp xỉ) [81], thuật toán này cung cấp bản ghi hiện tại cho 1 câu trả lời gần đúng trong 1 số hệ số nhân cố định của tối ưu (được gọi là tỷ lệ xấp xỉ α), sử dụng lập trình bán xác định & làm tròn ngẫu nhiên. Cụ thể, thuật toán GW đạt được tỷ lệ xấp xỉ được đảm bảo là $\alpha \sim 0.878$ cho đồ thị chung. Giới hạn dưới này có thể được nâng lên đối với các đồ thị cụ thể, ví dụ: đồ thị 3-chính quy không có trọng số trong đó $\alpha \sim 0.9326$ [82]. Việc triển khai thuật toán GW của chúng tôi dựa trên bộ giải CVXOPT nguồn mở, với CVXPY là giao diện mô hình. Đối với các đồ thị rất lớn có tới 1 triệu nút, các điểm chuẩn số không có sẵn, nhưng có thể so sánh giải pháp tốt nhất của chúng tôi $\text{cut}^\star$ với kết quả phân tích thu được trong [83], trong đó được hiển thị: với xác suất cao (trong giới hạn $n \to \infty$) kích thước của vết cắt cực đại cho đồ thị d-chính quy ngẫu nhiên với n nút được đưa ra bởi

$\text{cut}^\star = (\frac{d}{4} + P_* \sqrt{\frac{d}{4}} + O(\sqrt{d}))n + O(n)$. Ở đây $P_* \approx 0.7632$ đề cập đến 1 hằng số phổ quát liên quan đến năng lượng trạng thái cơ bản của mô hình Sherrington–Kirkpatrick [84, 85] có thể được biểu thị 1 cách phân tích thông qua công thức của Parisi [83]. Do đó, hãy lấy $\text{cut}_{\text{ub}} = \left(\frac{d}{4} + P_* \sqrt{\frac{d}{4}}\right) n$ làm ước lượng giới hạn trên cho kích thước cắt tối đa trong giới hạn n lớn. Bổ sung giới hạn trên này bằng giới hạn dưới đạt được bằng thuật toán xấp xỉ 0,5 ngẫu nhiên đơn giản (trung bình) cắt 1 nửa số cạnh, tạo ra kích thước cắt là $\text{cut}_{\text{rnd}} \approx \frac{d}{4} n$ cho đồ thị d-chính quy với $|E| = \frac{d}{2} n$. Kết quả của chúng tôi về kích thước cắt đạt được như 1 hàm của số đỉnh n được hiển thị trong Hình 4: Kết quả số cho MaxCut. Bảng bên trái; Kích thước cắt trung bình cho đồ thị chính quy d với $d = 3$ & $d = 5$ theo hàm số của số đỉnh n, trung bình bootstrap trên 20 trường hợp đồ thị ngẫu nhiên, cho cả phương pháp dựa trên GNN & thuật toán Goemans–Williamson (GW). Trên mỗi trường hợp đồ thị, bộ giải GNN được phép thực hiện tối đa 5 lần, & thuật toán GW thực hiện 100 lần. Đường liền cho $n \geq 10^3$ biểu diễn các giới hạn trên lý thuyết. Hình chèn: Tỷ lệ xấp xỉ tương đối ước tính được xác định là $\frac{\text{cut}^\star}{\text{cut}_{\text{ub}}}$ cho thấy: phương pháp của chúng tôi luôn đạt được các giải pháp chất lượng cao. Bảng bên phải: Thời gian chạy thuật toán tính bằng giây cho cả bộ giải GNN & thuật toán GW. Thanh lỗi đề cập đến độ lệch chuẩn khởi động 2 lần, lấy mẫu trên 20 trường hợp đồ thị ngẫu nhiên cho mọi điểm dữ liệu. Tất cả các kết quả đều là ước tính khởi động của giá trị trung bình, với thanh lỗi biểu thị độ lệch chuẩn khởi động 2 lần, lấy mẫu trên 20 đồ thị d-chính quy ngẫu nhiên cho mọi điểm dữ liệu. Đối với đồ thị có tối đa vài trăm nút, tìm: kiến trúc GCN 2 lớp đơn giản có thể hoạt động ngang bằng với thuật toán GW, đồng thời thể hiện lợi thế về thời gian chạy so với GW bắt đầu từ khoảng $n \approx 100$ nút. Đối với đồ thị lớn có $n \approx 10^4$ đến 10^6 nút, tìm: cách tiếp cận của chúng tôi luôn đạt được các giải pháp chất lượng cao với $\text{cut}^\star \gtrsim 0.9\text{cut}_{\text{ub}}$ cho cả $d = 3, d = 5$, tương ứng (i.e., tốt hơn nhiều so với bất kỳ thuật toán ngẫu nhiên ngây thơ nào). Như mong đợi đối với đồ thị d-chính quy, hãy tìm $\text{cut}^\star$ để chia tỷ lệ tuyến tính với số nút n, i.e., $\text{cut}^\star \approx \gamma_d n$ với $\gamma_3 \approx 1.28$ & $\gamma_5 \approx 1.93$ đối với $d = 3, d = 5$, tương ứng. Hơn nữa, khi sử dụng phần cứng GPU hiện đại, hãy quan sát khả năng mở rộng thời gian chạy thuận lợi ở kích thước hệ thống trung gian & lớn cho phép chúng ta giải quyết các trường hợp có $n = 10^6$ nút trong khoảng 10 phút (bao gồm cả các bước huấn luyện mô hình GNN & hậu xử lý). Cụ thể, như thể hiện trong Hình 4, hãy quan sát khả năng mở rộng thời gian chạy gần như tuyến tính với $\sim n$, đối với đồ thị d-chính quy lớn với $10^5 \leq n \leq 10^6$; trái ngược với việc quan sát tỷ lệ thuật toán GW là $\sim n^{3.5}$ đối với kích thước vấn đề trong phạm vi $n \lesssim 250$, do đó cho thấy độ phức tạp thời gian (dự kiến) $\tilde{O}(n^{3.5})$ của phương pháp điểm bên trong (thường được sử dụng để giải chương trình bán xác định cơ bản của thuật toán GW) chi phối thời gian chạy thuật toán GW [86, 87].

. . .

Maximum Independent Set. MIS problem is a prominent combinatorial optimization problem with practical applications in network design [93] & finance [94], & is closely related to maximum clique, minimum vertex cover, & set packing problems. In quantum community, MIS problem has recently attracted significant interest [95] as a potential target use case for novel experimental platforms based on neutral atom arrays [96]. MIS problem reads as follows. Given an undirected graph $G = (V, E)$, an independent set is a subset of vertices that are not connected with each other. MIS problem is then task to find largest independent set, with its (maximum) cardinality typically denoted as independence number α. To formulate MIS problem . . .

- 6. Applications in industry. While our previous analysis has focused on canonical graph optimization problems e.g. maximum cut & maximum independent set, in this section discuss real-world applications in industry for which our solver could provide solutions, in particular at potentially unprecedented problem scales. Focus on applications of QUBO formalism, even though our methodology is not limited to this modeling framework. 1st review existing literature, providing relevant references across a wide stack of problem domains. Thereafter, explicitly show how to distill combinatorial QUBO problems for a few select real-world use cases, from risk diversification in finance to sensor placement problems in water distribution networks. Once in QUBO format, problem can be plugged into our general-purpose physics-inspired GNN solver, as outlined above.

  – Trong khi phân tích trước đây của chúng tôi tập trung vào các bài toán tối ưu hóa đồ thị chuẩn, ví dụ như cắt cực đại & tập độc lập cực đại, phần này thảo luận về các ứng dụng thực tế trong công nghiệp mà bộ giải của chúng tôi có thể cung cấp giải pháp, đặc biệt là ở các quy mô bài toán chưa từng có tiền lệ. Tập trung vào các ứng dụng của hình thức QUBO, mặc dù phương pháp luận của chúng tôi không bị giới hạn trong khuôn khổ mô hình này. Trước tiên, hãy xem xét các tài liệu hiện có, cung cấp các tài liệu tham khảo liên quan trên nhiều lĩnh vực bài toán. Sau đó, hãy trình bày rõ ràng cách chắt lọc các bài toán QUBO tổ hợp cho 1 số trường hợp sử dụng thực tế được chọn lọc, từ phân tán rủi ro trong tài chính đến các bài toán đặt cảm biến trong mạng lưới phân phối nước. Khi đã ở định dạng QUBO, bài toán có thể được đưa vào bộ giải GNN lấy cảm hứng từ vật lý đa năng của chúng tôi, như đã nêu ở trên.

  As extensively reviewed in [1,2,66], QUBO (or, equivalently, Ising) formalism provides a comprehensive modeling framework encompassing a vast array of optimization problems, including knapsack problems, task (resource) allocation problems, & capital budgeting problems, among others. Specifically, applicability of QUBO representation has been reported for problem settings involving circuit board layouts [100], capital budgeting in financial analysis [101], computer aided design (CAD) [102], electronic traffic management [103, 104], cellular radio channel allocation [105], molecular conformation [106], & prediction of epileptic seizures [107], among others. Practical applications of MaxCut problem can be found in machine scheduling [71], image recognition [72], & electronic circuit layout design [73]. Similarly, in what follows discuss in detail 3 select use cases & how they can be cast in QUBO form & thus made amenable to our solver.

  – Như đã được xem xét rộng rãi trong [1,2,66], hình thức QUBO (hoặc tương đương là Ising) cung cấp 1 khuôn khổ mô hình hóa toàn diện bao gồm 1 loạt lớn các vấn đề tối ưu hóa, bao gồm các vấn đề ba lô, các vấn đề phân bổ nhiệm vụ (nguồn lực), & các vấn đề ngân sách vốn, trong số những vấn đề khác. Cụ thể, khả năng áp dụng biểu diễn QUBO đã được báo cáo cho các thiết lập vấn đề liên quan đến bố trí bảng mạch [100], ngân sách vốn trong phân tích tài chính [101], thiết kế hỗ trợ máy tính (CAD) [102], quản lý lưu lượng điện tử [103, 104], phân bổ kênh vô tuyến di động [105], cấu hình phân tử [106], & dự

đoán các cơn động kinh [107], trong số những vấn đề khác. Các ứng dụng thực tế của vấn đề MaxCut có thể được tìm thấy trong lập lịch máy [71], nhận dạng hình ảnh [72], & thiết kế bố trí mạch điện tử [73]. Tương tự như vậy, trong phần sau sẽ thảo luận chi tiết về 3 trường hợp sử dụng được chọn & cách chúng có thể được đúc ở dạng QUBO & do đó phù hợp với trình giải của chúng tôi.

. . .

- 7. Conclusion & outlook.

3 GNNs for Computer Music

3.1 Matej Bevec, Marko Tkalčič, Matevž Pesek. Hybrid Music Recommendation with Graph Neural Networks

- Abstract. Modern music streaming services rely on recommender systems to help users navigate within their large collections. Collaborative filtering (CF) methods, that leverage past user-item interactions, have been most successful, but have various limitations, like performing poorly among sparsely connected items. Conversely, content-based models circumvent data-sparsity issue by recommending based on item content alone, but have seen limited success. Recently, graph-based ML approaches have shown, in other domains, to be able to address aforementioned issues. GNN in particular promise to learn from both complex relationships within a user interaction graph, as well as content to generate hybrid recommendations. Here propose a music recommender system using a state-of-art GNN, PinSage, & evaluate it on a novel Spotify dataset against traditional CF, graph-based CF & content-based methods on a related song prediction task, venturing beyond accuracy in our evaluation. Our experiments show: (i) our approach is among top performers & stands out as most well rounded compared to baselines, (ii) graph-based CF methods outperform matrix-based CF approaches, suggesting: user interaction data may be better represented as a graph & (iii) in our evaluation, CF methods do not exhibit a performance drop in long tail, where hybrid approach does not offer an advantage.

  – Tóm tắt. Các dịch vụ phát nhạc trực tuyến hiện đại dựa vào hệ thống đề xuất để giúp người dùng điều hướng trong các bộ sưu tập lớn của họ. Các phương pháp lọc cộng tác (CF), tận dụng các tương tác giữa người dùng & mục trong quá khứ, đã thành công nhất, nhưng có nhiều hạn chế, chẳng hạn như hoạt động kém giữa các mục được kết nối thưa thớt. Ngược lại, các mô hình dựa trên nội dung tránh được vấn đề thưa thớt dữ liệu bằng cách đề xuất chỉ dựa trên nội dung mục, nhưng đạt được thành công hạn chế. Gần đây, các phương pháp học máy dựa trên đồ thị đã cho thấy, trong các lĩnh vực khác, có thể giải quyết các vấn đề đã đề cập ở trên. Đặc biệt, GNN hứa hẹn sẽ học hỏi từ cả các mối quan hệ phức tạp trong biểu đồ tương tác của người dùng, cũng như nội dung để tạo ra các đề xuất kết hợp. Ở đây, đề xuất 1 hệ thống đề xuất âm nhạc sử dụng GNN hiện đại, PinSage, & đánh giá nó trên 1 tập dữ liệu Spotify mới so với CF truyền thống, CF dựa trên đồ thị & các phương pháp dựa trên nội dung trên 1 tác vụ dự đoán bài hát có liên quan, vượt ra ngoài độ chính xác trong đánh giá của chúng tôi. Các thí nghiệm của chúng tôi cho thấy: (i) phương pháp của chúng tôi nằm trong số những phương pháp có hiệu suất cao nhất & nổi bật là toàn diện nhất so với các phương pháp cơ sở, (ii) các phương pháp CF dựa trên đồ thị vượt trội hơn các phương pháp CF dựa trên ma trận, cho thấy: dữ liệu tương tác của người dùng có thể được biểu diễn tốt hơn dưới dạng đồ thị & (iii) trong đánh giá của chúng tôi, các phương pháp CF không cho thấy sự sụt giảm hiệu suất trong phần đuôi dài, trong khi phương pháp kết hợp không mang lại lợi thế.

Keywords. embeddings, music recommendation systems, graph neural networks, PinSage, beyond-accuracy evaluation.

- 1. Introduction. In recent years, a large share of music consumption has moved to subscription-based streaming platforms e.g. Spotify, Apple Music, & YouTube Music. With increasingly large catalogs of songs provided by these services, recommender systems (RS) play a crucial role of matchmaker between content & users.

  – Trong những năm gần đây, 1 phần lớn nhu cầu tiêu thụ âm nhạc đã chuyển sang các nền tảng phát trực tuyến trả phí như Spotify, Apple Music & YouTube Music. Với danh mục bài hát ngày càng phong phú do các dịch vụ này cung cấp, hệ thống đề xuất (RS) đóng vai trò quan trọng trong việc kết nối nội dung & người dùng.

  Regardless of domain, recommender systems have traditionally been formulated as models that aim to predict which items different users might prefer. In this case, recommendation refers to predicting unknown entries in a user-item rating matrix R where each entry r_{ij} denotes how user i rates item j. Modern recommender systems, however, are wide in scope & deal with many more related tasks. In music domain, these tasks include feed recommendation, playlist generation, related song recommendation, etc.

  – Bất kể lĩnh vực nào, các hệ thống đề xuất theo truyền thống được xây dựng dưới dạng các mô hình nhằm mục đích dự đoán những mục nào mà người dùng khác nhau có thể thích. Trong trường hợp này, đề xuất đề cập đến việc dự đoán các mục chưa biết trong ma trận đánh giá người dùng-mục R, trong đó mỗi mục r_{ij} biểu thị cách người dùng i đánh giá mục j. Tuy nhiên, các hệ thống đề xuất hiện đại có phạm vi rộng & xử lý nhiều tác vụ liên quan hơn. Trong lĩnh vực âm nhạc, các tác vụ này bao gồm đề xuất nguồn cấp dữ liệu, tạo danh sách phát, đề xuất bài hát liên quan, v.v.

  Most widely adopted & successful category of methods used in recommender systems is collaborative filtering (CF). CF algorithms operate exclusively on user behavior data in form of interaction matrix R, drawing from assumption that similar users prefer similar items (Xiaoyuan & Khoshgoftaar 2009). These methods are still among top performers in terms of recommendation accuracy. They also carry many favorable characteristics, e.g. domain-independence, since no knowledge about content, only user data, is needed. They do, however, suffer from some crucial limitations. One is data sparsity problem, which refers

to fact: CF algorithms cannot provide reliable recommendations songs with insufficient (sparse) interaction data – a situation that is particularly common in music catalogs.

– Loại phương pháp được áp dụng rộng rãi & thành công nhất trong các hệ thống đề xuất là lọc cộng tác (CF). Các thuật toán CF hoạt động hoàn toàn trên dữ liệu hành vi người dùng dưới dạng ma trận tương tác R, dựa trên giả định rằng những người dùng tương tự sẽ thích các mục tương tự (Xiaoyuan & Khoshgoftaar 2009). Các phương pháp này vẫn nằm trong số những phương pháp có hiệu suất cao nhất về độ chính xác đề xuất. Chúng cũng mang nhiều đặc điểm thuận lợi, ví dụ như độc lập với miền, vì không cần kiến thức về nội dung, chỉ cần dữ liệu người dùng. Tuy nhiên, chúng cũng gặp phải 1 số hạn chế quan trọng. 1 trong số đó là vấn đề thưa thớt dữ liệu, ám chỉ thực tế: các thuật toán CF không thể cung cấp các đề xuất đáng tin cậy cho các bài hát khi dữ liệu tương tác không đủ (thưa thớt) – 1 tình huống đặc biệt phổ biến trong các danh mục âm nhạc.

A different approach to recommendations is content-based filtering (CBF). CBF methods compute similarities between items or users based solely on their content, ignoring user-item interactions. In field of music recommendations systems (MRS), "content" in CBF usually refers to metadata e.g. genre, artist information or lyrics (Lops et al. 2011; Schedl 2019), while music information retrieval (MIR) uses "content" to refer to raw music audio & has produced many models that extract features from audio to effectively solve tasks, e.g. genre classification, emotion detection or instrument recognition (Kim et al. 2010; Cramer et al. 2019; Choi et al. 2018). Such approaches do not suffer from cold-start & data-sparsity issues but in general give less accurate recommendations. Metadata tends to be too broad to sufficiently model user taste (e.g., genre) & gives subpar recommendations. Conversely, music audio itself is a rich source of information with potential to directly recommend music that is perceptually in line with user's taste. Research in this domain, however, faces semantic gap – a substantial chasm between raw audio signal & perceptual music characteristics that affect human preference Celma Herrada et al. 2006). As such, direct application of CBF to music recommender systems has been limited.

– 1 cách tiếp cận khác đối với các đề xuất là lọc dựa trên nội dung (CBF). Các phương pháp CBF tính toán điểm tương đồng giữa các mục hoặc người dùng chỉ dựa trên nội dung của chúng, bỏ qua tương tác giữa người dùng & mục. Trong lĩnh vực hệ thống đề xuất âm nhạc (MRS), "nội dung" trong CBF thường đề cập đến siêu dữ liệu, ví dụ như thể loại, thông tin nghệ sĩ hoặc lời bài hát (Lops & cộng sự, 2011; Schedl, 2019), trong khi truy xuất thông tin âm nhạc (MIR) sử dụng "nội dung" để chỉ âm thanh nhạc thô & đã tạo ra nhiều mô hình trích xuất các tính năng từ âm thanh để giải quyết hiệu quả các tác vụ, ví dụ như phân loại thể loại, phát hiện cảm xúc hoặc nhận dạng nhạc cụ (Kim & cộng sự, 2010; Cramer & cộng sự, 2019; Choi & cộng sự, 2018). Các cách tiếp cận như vậy không gặp phải các vấn đề về khởi động lạnh & dữ liệu thưa thớt nhưng nhìn chung đưa ra các đề xuất kém chính xác hơn. Siêu dữ liệu có xu hướng quá rộng để mô hình hóa đầy đủ sở thích của người dùng (ví dụ: thể loại) & đưa ra các đề xuất kém chất lượng. Ngược lại, bản thân âm nhạc là 1 nguồn thông tin phong phú với tiềm năng đề xuất trực tiếp âm nhạc phù hợp với thị hiếu của người dùng. Tuy nhiên, nghiên cứu trong lĩnh vực này đang gặp phải khoảng cách ngữ nghĩa - 1 khoảng cách đáng kể giữa tín hiệu âm thanh thô & các đặc điểm âm nhạc cảm nhận được, ảnh hưởng đến sở thích của con người (Celma Herrada & cộng sự, 2006). Do đó, việc áp dụng trực tiếp CBF vào các hệ thống đề xuất âm nhạc còn hạn chế.

Limitations of pure CF & CBF have lead researchers in direction of hybrid recommender systems, which leverage both interaction data & content in an attempt to mitigate shortcomings of exclusively using one or the other. This can be done in various ways, e.g. combining features learned from CF with content-based features (Vall et al. 2019) or by using content to guide CF process (Liang et al. 2015).

– Những hạn chế của CF thuần túy & CBF đã dẫn dắt các nhà nghiên cứu đến các hệ thống đề xuất lai, tận dụng cả dữ liệu tương tác & nội dung để giảm thiểu những hạn chế của việc chỉ sử dụng 1 trong hai. Điều này có thể được thực hiện theo nhiều cách, ví dụ: kết hợp các đặc điểm học được từ CF với các đặc điểm dựa trên nội dung (Vall & cộng sự, 2019) hoặc bằng cách sử dụng nội dung để hướng dẫn quy trình CF (Liang & cộng sự, 2015).

Another rapidly growing branch of ML is ML with graphs. Graphs have always been a highly expressive way to represent relational data, but only recent advances in field have managed to truly utilize these structures in a ML context. Spearheaded by GNN – DL architectures that operate directly on graphs – graph-based methods now provide better solutions to relational tasks e.g. predicting protein–protein interactions, physical systems modeling or suggesting friends in social networks (Hamilton et al. 2017b; Wu et al. 2021).

– 1 nhánh phát triển nhanh chóng khác của ML là ML với đồ thị. Đồ thị luôn là 1 cách biểu diễn dữ liệu quan hệ rất biểu cảm, nhưng chỉ những tiến bộ gần đây trong lĩnh vực này mới thực sự tận dụng được các cấu trúc này trong bối cảnh ML. Dẫn đầu bởi kiến trúc GNN – DL hoạt động trực tiếp trên đồ thị – các phương pháp dựa trên đồ thị hiện cung cấp các giải pháp tốt hơn cho các tác vụ quan hệ, ví dụ như dự đoán tương tác protein-protein, mô hình hóa hệ thống vật lý hoặc gợi ý bạn bè trên mạng xã hội (Hamilton & cộng sự, 2017b; Wu & cộng sự, 2021).

In present study, aim to inspect utility of this new family of methods in context of music recommendation, by applying them to a fundamental MRS task of *related song recommendation* (i.e., prediction of ground-truth similar pairs of songs). Consider this is a natural 1st task since a good model of similarity can be, & often is, basis for various more complex tasks.

– Trong nghiên cứu hiện tại, mục tiêu là kiểm tra tính hữu dụng của nhóm phương pháp mới này trong bối cảnh đề xuất âm nhạc, bằng cách áp dụng chúng vào nhiệm vụ MRS cơ bản là *đề xuất bài hát liên quan* (i.e., dự đoán các cặp bài hát tương tự trên thực tế). Hãy coi đây là nhiệm vụ đầu tiên tự nhiên vì 1 mô hình tương đồng tốt có thể, & thường là, cơ sở cho nhiều nhiệm vụ phức tạp hơn.

Considering success of graph-based approaches on many tasks which deal with relational data, our research question: *Can ML*

with graphs, & GNN in particular, address shortcomings of previous CF & CBF approaches in context of music recommendation?

– Xem xét thành công của các phương pháp tiếp cận dựa trên đồ thị trên nhiều nhiệm vụ liên quan đến dữ liệu quan hệ, câu hỏi nghiên cứu của chúng tôi: *liệu ML với đồ thị, & GNN nói riêng, có thể giải quyết những thiếu sót của các phương pháp CF & CBF trước đây trong bối cảnh đề xuất âm nhạc không?*

Music recommendation can be translated to graph domain by representing a user–song matrix or a set of user-created playlists as a bipartite graph of songs & users or playlists, where links denote interaction or membership. By translating user interaction information to graph topology, graph-based methods can model complex relations between items &/or users that matrix-based CF approaches might struggle with. *Therefore conjecture: both hybrid methods (GNN) & graph-based CF methods outperform traditional matrix-based approaches.*

– Đề xuất âm nhạc có thể được chuyển đổi sang miền đồ thị bằng cách biểu diễn ma trận bài hát - người dùng hoặc 1 tập hợp danh sách phát do người dùng tạo dưới dạng đồ thị 2 phần gồm bài hát & người dùng hoặc danh sách phát, trong đó các liên kết biểu thị tương tác hoặc thành viên. Bằng cách chuyển đổi thông tin tương tác của người dùng sang cấu trúc đồ thị, các phương pháp dựa trên đồ thị có thể mô hình hóa các mối quan hệ phức tạp giữa các mục &/hoặc người dùng mà các phương pháp CF dựa trên ma trận có thể gặp khó khăn. *Do đó, có thể suy đoán: cả phương pháp lai (GNN) & phương pháp CF dựa trên đồ thị đều vượt trội hơn các phương pháp dựa trên ma trận truyền thống.*

Additionally, & unlike other hybrid approaches, GNNs are also intrinsically hybrid – they form node embeddings by aggregating neighboring nodes' features based on graph topology. I.e., nodes representing songs can be associated with audio embeddings to produce hybrid song representations, e.g. *Therefore conjecture: hybrid method, GNN, provides more accurate recommendation than CBF, while being more successful than CF in handling sparse data.*

– Ngoài ra, & không giống như các phương pháp lai khác, GNN cũng mang tính lai về bản chất – chúng tạo ra các nhúng nút bằng cách tổng hợp các đặc điểm của các nút lân cận dựa trên cấu trúc đồ thị. E.g., các nút biểu diễn bài hát có thể được liên kết với các nhúng âm thanh để tạo ra các biểu diễn bài hát lai, ví dụ: *Do đó, có thể suy đoán: phương pháp lai, GNN, cung cấp khuyến nghị chính xác hơn CBF, đồng thời thành công hơn CF trong việc xử lý dữ liệu thưa thớt.*

While accuracy is how recommenders are usually accessed, it has been pointed out by many: these metrics do not reflect all characteristics of recommendations that may be relevant in a real-world setting. Following this rationale, also study our results from perspective of beyond-accuracy objectives & *conjecture: looking beyond accuracy will provide an alternative picture in terms of ranking methods' performance.*

– Mặc dù độ chính xác thường là cách tiếp cận người đề xuất, nhưng nhiều người đã chỉ ra rằng: các số liệu này không phản ánh tất cả các đặc điểm của đề xuất có thể liên quan trong bối cảnh thực tế. Dựa trên cơ sở lý luận này, hãy nghiên cứu kết quả của chúng tôi từ góc độ các mục tiêu vượt ra ngoài độ chính xác & *giả thuyết: nhìn xa hơn độ chính xác sẽ cung cấp 1 bức tranh khác về hiệu suất của các phương pháp xếp hạng.*

In present study, apply a state-of-art GNN method, PinSage (Ying et al. 2018) & other graph-based approaches to public Spotify data to explore utility of this paradigm in space of hybrid music recommendation. Our contributions can be summarized as follows:

- Implementation of a state-of-art GNN algorithm (PinSage) in role of a hybrid music recommender, including an ablation study, & as such, an application of ML with graphs to field of MRS.
- An evaluation, in terms of recommendation accuracy, of various graph-based methods on real & current music consumption data against content-based methods (CBF) & user-data-based methods, including traditional CF, as well as graph-based approaches.
- An additional analysis of considered methods in terms of *beyond-accuracy objectives*, with an aim to quantify characteristics of recommender systems in a more holistic way.

– Trong nghiên cứu này, chúng tôi áp dụng phương pháp GNN tiên tiến, PinSage (Ying & cộng sự, 2018) & các phương pháp tiếp cận dựa trên đồ thị khác vào dữ liệu Spotify công khai để khám phá tính hữu dụng của mô hình này trong lĩnh vực đề xuất âm nhạc kết hợp. Đóng góp của chúng tôi có thể được tóm tắt như sau:

- Triển khai thuật toán GNN tiên tiến (PinSage) trong vai trò là công cụ đề xuất âm nhạc kết hợp, bao gồm nghiên cứu cắt bỏ, & do đó, ứng dụng ML với đồ thị vào lĩnh vực MRS.
- Đánh giá, về độ chính xác của đề xuất, các phương pháp dựa trên đồ thị khác nhau trên dữ liệu tiêu thụ âm nhạc thực tế & hiện tại so với các phương pháp dựa trên nội dung (CBF) & các phương pháp dựa trên dữ liệu người dùng, bao gồm CF truyền thống, cũng như các phương pháp dựa trên đồ thị.
- Phân tích bổ sung về các phương pháp được xem xét theo *các mục tiêu vượt ra ngoài độ chính xác*, nhằm mục đích định lượng các đặc điểm của hệ thống đề xuất 1 cách toàn diện hơn.

To aid further research on intersection between ML with graphs & MRS, publish full experimental code https://github.com/MatejBevec/gcn-song-embeddings, including PinSage implementation. Also release our novel dataset, a large-scale bipartite playlist-song membership graph, which name *Spotify Graph.*

– Để hỗ trợ nghiên cứu sâu hơn về giao điểm giữa ML với đồ thị & MRS, hãy công bố toàn bộ mã thử nghiệm https://github.com/MatejBevec/gcn-song-embeddings, bao gồm cả triển khai PinSage. Đồng thời, chúng tôi cũng phát hành bộ dữ liệu mới của mình, 1 đồ thị thành viên bài hát-danh sách phát 2 phần quy mô lớn, có tên là *Spotify Graph.*

- **2. Related work.** As our study builds on previous work in both recommender systems & graph-based ML, survey both & position our work in this landscape.

  – Vì nghiên cứu của chúng tôi dựa trên công trình trước đây về cả hệ thống đề xuất & ML dựa trên đồ thị, hãy khảo sát cả & định vị công trình của chúng tôi trong bối cảnh này.

  - 2.1. Music recommendation: tasks & challenges. Music recommender systems (MRS) deal with similar tasks as all recommender systems (RS). However, due to particularities of music as a medium, e.g. short duration of items, sequential consumption, scale of collections, or a highly subjective & variable listening intent, they face some unique challenges.

    – Đề xuất âm nhạc: nhiệm vụ & thách thức. Hệ thống đề xuất âm nhạc (MRS) xử lý các nhiệm vụ tương tự như tất cả các hệ thống đề xuất (RS). Tuy nhiên, do đặc thù của âm nhạc như 1 phương tiện, ví dụ như thời lượng ngắn của các mục, mức độ tiêu thụ tuần tự, quy mô bộ sưu tập hoặc ý định nghe nhạc rất chủ quan & biến động, chúng phải đối mặt với 1 số thách thức riêng.

- 3. Theoretical framework.

  - 3.1. Related song recommendation task. Since our study explores utility of graph-based methods in MRS from a broad perspective, devise our experiment around 1 of simplest yet most fundamental tasks in music recommendation – *similar song prediction*. I.e., finding similar music to a single seed (query) song, without any input about user or context. This has advantage of both being able to be framed in language of vastly different algorithms we test, & of being widely applicable to more complex tasks. E.g., it can be used for playlist completion by fetching similar songs to existing set with some randomness.

    – Nhiệm vụ đề xuất bài hát liên quan. Vì nghiên cứu của chúng tôi khám phá tính hữu dụng của các phương pháp dựa trên đồ thị trong MRS từ góc nhìn rộng, hãy thiết kế thử nghiệm của chúng tôi xoay quanh 1 trong những nhiệm vụ đơn giản nhưng cơ bản nhất trong đề xuất âm nhạc – *dự đoán bài hát tương tự*. Cụ thể là tìm kiếm những bài hát tương tự với 1 bài hát gốc (truy vấn) duy nhất, mà không cần bất kỳ thông tin đầu vào nào về người dùng hoặc ngữ cảnh. Điều này có lợi thế là có thể được diễn đạt bằng ngôn ngữ của các thuật toán rất khác nhau mà chúng tôi thử nghiệm, & có thể áp dụng rộng rãi cho các nhiệm vụ phức tạp hơn. Ví dụ: nó có thể được sử dụng để hoàn thành danh sách phát bằng cách tìm nạp các bài hát tương tự vào bộ bài hát hiện có với 1 số tính ngẫu nhiên.

    Opt to use *consecutive song plays in listening sessions* ax proxy for similarity. More specifically, assume: songs q, i should be considered similar, if they often appear together in listening sessions. While this may not be an ideal analogy, it is data we find available & is inspired by PinSage paper (Ying et al. 2018).

    – Chọn sử dụng *bài hát được phát liên tiếp trong các buổi nghe* ax proxy cho tính tương đồng. Cụ thể hơn, giả sử: các bài hát q, i nên được coi là tương đồng, nếu chúng thường xuất hiện cùng nhau trong các buổi nghe. Mặc dù đây có thể không phải là phép so sánh lý tưởng, nhưng đây là dữ liệu chúng tôi tìm thấy & được lấy cảm hứng từ bài báo của PinSage (Ying & cộng sự, 2018).

    Algorithms do have access to another source of information, playlist membership graph, but only in training. This is not part of evaluation & can be seen as background information. This choice, as well, as partially inspired by experiment design in PinSage study. Below is a theoretical definition of proposed experiment. Consider following data:

    1. A playlist-song graph G, represented as adjacency matrix $G \in \mathbb{R}^{n \times m}$, where each entry g_{ip} is a Boolean value, indicating whether song i is present in playlist p.
    2. Audio-content features F for each song, represented as matrix $F \in \mathbb{R}^{n \times d}$, where row F_i is a d-dimensional dense embedding, representing 30s audio excerpt of song i.
    3. A set of song co-occurrences P, consecutive song plays from users' listening sessions. Name these "positive pairs" & represent them as matrix $P \in \mathbb{R}^{n \times n}$ where p_{ij} counts number of times song i, j appear in such a pair.

    Our related song recommendation task is then defined as follows: **Predict song co-occurrences in P, i.e., given a pair $(q, i) : P_{q,i} > 0$, an algorithm should include i among its recommendations r_q to query q.**

    – Các thuật toán có thể truy cập vào 1 nguồn thông tin khác, biểu đồ thành viên danh sách phát, nhưng chỉ trong quá trình đào tạo. Đây không phải là 1 phần của quá trình đánh giá & có thể được xem là thông tin cơ bản. Lựa chọn này cũng được lấy cảm hứng 1 phần từ thiết kế thử nghiệm trong nghiên cứu của PinSage. Dưới đây là định nghĩa lý thuyết về thử nghiệm được đề xuất. Xem xét dữ liệu sau:

    1. Đồ thị danh sách phát-bài hát G, được biểu diễn dưới dạng ma trận kề $G \in \mathbb{R}^{n \times m}$, trong đó mỗi mục g_{ip} là 1 giá trị Boolean, cho biết bài hát i có trong danh sách phát p hay không.
    2. Đặc trưng nội dung âm thanh F cho mỗi bài hát, được biểu diễn dưới dạng ma trận $F \in \mathbb{R}^{n \times d}$, trong đó hàng F_i là 1 nhúng dày đặc d chiều, biểu diễn đoạn trích âm thanh 30 giây của bài hát i.
    3. 1 tập hợp các bài hát đồng xuất hiện P, được phát liên tiếp từ các phiên nghe nhạc của người dùng. Đặt tên cho các "cặp dương" này & biểu diễn chúng dưới dạng ma trận $P \in \mathbb{R}^{n \times n}$, trong đó p_{ij} đếm số lần bài hát i, j xuất hiện trong 1 cặp như vậy.

    Nhiệm vụ đề xuất bài hát liên quan của chúng ta sau đó được định nghĩa như sau: **Dự đoán các bài hát đồng xuất hiện trong P, tức là, với 1 cặp $(q, i) : P_{q,i} > 0$, thuật toán nên bao gồm i trong số các đề xuất r_q của nó để truy vấn q.**

    To this end, all tested methods utilize 1 or more of above data sources to produce a similarity function $sim(i, j)$ between songs i, j. Embedding methods 1st produce dense embedding $V \in \mathbb{R}^{n \times d}$ & compute $sim(i, j)$ as cosine similarity between

2 embedding vectors $cosine(V_i, V_j)$. Recommendations to q are then obtained by querying k nearest neighbors to q w.r.t. sim. Sect. 4 for information about particular data & methods used.

– Để đạt được mục đích này, tất cả các phương pháp đã thử nghiệm đều sử dụng 1 hoặc nhiều nguồn dữ liệu trên để tạo ra hàm tương tự $sim(i, j)$ giữa các bài hát i, j. Các phương pháp nhúng đầu tiên tạo ra nhúng dày đặc $V \in \mathbb{R}^{n \times d}$ & tính $sim(i, j)$ dưới dạng độ tương tự cosin giữa 2 vectơ nhúng $cosine(V_i, V_j)$. Sau đó, các khuyến nghị cho q được thu thập bằng cách truy vấn k lân cận gần nhất của q so với sim. Xem Mục 4 để biết thông tin về dữ liệu cụ thể & các phương pháp được sử dụng.

- 3.2. PinSage. At its core, PinSage is a GCN-based hybrid node embedding algorithm, which, compared to previous GCN approaches, vastly improves scalability of system, as well as quality of learned embeddings. This is mainly due to a new way of computing convolutions & an improved training procedure. In our work, follow (Ying et al. 2018) to implement a simplified version of the algorithm. See PinSage as a state-of-art GCN methods & a reasonable choice to explore potential of hybrid graph-based methods in music recommendation. As such, primarily interested in its promises of improved recommendation quality, rather than scalability needed at production scale. Following sects briefly describe model architecture & training procedure. If this is of no interest, advise reader to skip Sect. 4.

4 GNNs for Scheduling Problems

4.1 Pablo Arino Fernandez. Solving the Job Shop Scheduling Problem with Graph Neural Networks: A Customizable Reinforcement Learning Environment. Bachelor Thesis. 2025

- Abstract. Job shop scheduling problem is an NP-hard combinatorial optimization problem relevant to manufacturing & timetable scheduling. Because of problem's combinatorial nature, heuristics are typically used. They aim to provide "good enough" solutions. 1 of most common approaches: use priority dispatching rules. These rules decide which tasks should be prioritized based on simple heuristics, e.g. choosing the one that finishes earliest.

  – Bài toán lập lịch xưởng là 1 bài toán tối ưu hóa tổ hợp NP-khó liên quan đến lập lịch sản xuất & thời gian biểu. Do tính chất tổ hợp của bài toán, các phương pháp tìm kiếm (heuristic) thường được sử dụng. Chúng nhằm mục đích cung cấp các giải pháp "đủ tốt". 1 trong những cách tiếp cận phổ biến nhất: sử dụng các quy tắc phân bổ ưu tiên. Các quy tắc này quyết định những tác vụ nào nên được ưu tiên dựa trên các phương pháp tìm kiếm đơn giản, ví dụ: chọn tác vụ hoàn thành sớm nhất.

  In recent years, there have been attempts to replace these rules with DL models, which learn to assign priorities through data. Specifically, graph neural networks have emerged as an alternative. However, numerous factors can be customized when generating training data: graph used to represent problem, features used to represent each node, definition of tasks available for dispatching at each moment, & reward function used. Due to lack of modular libraries that allow choosing & experimenting with new components, training such models requires significant investments in development time.

  – Trong những năm gần đây, đã có những nỗ lực thay thế các quy tắc này bằng các mô hình DL, học cách gán mức độ ưu tiên thông qua dữ liệu. Cụ thể, mạng nơ-ron đồ thị đã nổi lên như 1 giải pháp thay thế. Tuy nhiên, nhiều yếu tố có thể được tùy chỉnh khi tạo dữ liệu huấn luyện: đồ thị được sử dụng để biểu diễn bài toán, các đặc trưng được sử dụng để biểu diễn từng nút, định nghĩa các tác vụ có thể phân bổ tại mỗi thời điểm, & hàm thưởng được sử dụng. Do thiếu các thư viện mô-đun cho phép lựa chọn & thử nghiệm các thành phần mới, việc huấn luyện các mô hình như vậy đòi hỏi đầu tư đáng kể vào thời gian phát triển.

  This work introduces a new modular library `JobShopLib` that incorporates a reinforcement learning environment that allows customizing each of these factors & creating new ones. Furthermore, several task selection models have been trained through imitation learning using this library's environment to show its utility. 1 of these models outperformed various graph-based dispatchers while considering only individual task features. This suggests that aspects e.g. node features are important, further justifying need for customization that our library provides. Moreover, our graph neural network-based model has achieved very close to state-of-art results in large-scale problems. Both results suggest there is still significant room for improvement in developing these types of models. `JobShopLib` with its environments for generating training data, aims to provide tools for such future experimentation.

  – Công trình này giới thiệu 1 thư viện mô-đun mới `JobShopLib` kết hợp môi trường học tăng cường cho phép tùy chỉnh từng yếu tố này & tạo ra những yếu tố mới. Hơn nữa, 1 số mô hình lựa chọn tác vụ đã được đào tạo thông qua học mô phỏng bằng cách sử dụng môi trường của thư viện này để chứng minh tính hữu ích của nó. 1 trong số các mô hình này đã vượt trội hơn nhiều bộ phân phối dựa trên đồ thị trong khi chỉ xem xét các tính năng tác vụ riêng lẻ. Điều này cho thấy các khía cạnh như tính năng nút rất quan trọng, càng chứng minh thêm nhu cầu tùy chỉnh mà thư viện của chúng tôi cung cấp. Hơn nữa, mô hình dựa trên mạng nơ-ron đồ thị của chúng tôi đã đạt được kết quả rất gần với công nghệ tiên tiến trong các vấn đề quy mô lớn. Cả 2 kết quả đều cho thấy vẫn còn nhiều chỗ để cải thiện trong việc phát triển các loại mô hình này. `JobShopLib` với môi trường tạo dữ liệu đào tạo của nó nhằm mục đích cung cấp các công cụ cho các thử nghiệm trong tương lai như vậy.

- List of Acronyms. JSSP Job Shop Scheduling Problem, PDRs Priority Dispatching Rules, SPT Shortest Processing Time, MWKR Most Work Remaining. FCFS First Come First Served, MOR Most Operations Remaining, MILP Mixed Integer Linear Programming, CP Constraint Programming, NNs Neural Networks, tanh hyperbolic tangent, ReLU Rectified Linear Unit, ELU Exponential Linear Unit, SGD Stochastic Gradient Descent, ANNs Artificial Neural Networks, GNNs Graph Neural Networks, MP Message Passing, RGNNs Relational Graph Neural Networks, GCNs Graph Convolutional Networks, GATs

Graph Attention Networks, RGATv2 Relational Graph Attention Network v2, GINs Graph Isomorphism Networks, RGIN Relational Graph Isomorphism Network, MLP Multilayer Perceptron, MDP Markov Decision Process, SMDP Semi-Markov Decision Process, RL Reinforcement Learning, IL Imitation Learning, BC Behavioral Cloning, A2C Advantage Actor-Critic, PPO Proximal Policy Optimization, PyG Pytorch Geometric

- Notation. Job-Shop-Scheduling-Problem-related notation:

1. $\mathcal{J}$: set of jobs
2. $\mathcal{M}$: set of machines
3. $|\mathcal{J}| \times |\mathcal{M}|$: problem size (number of jobs $\times$ number of machines)
4. J_i: job i in set $\mathcal{J}$
5. O_{ij}: operation j of job i
6. M_j: machine j in set $\mathcal{M}$
7. p_{ij}: processing time of operation O_{ij} (also called duration)
8. S_{ij}: start time of operation O_{ij}
9. C_{ij}: completion time of operation O_{ij} ($C_{ij} = S_{ij} + p_{ij}$)
10. $C_{\max}$: makespan (maximum completion time of all operations)

Graph-Neural-Network-related notation:

1. $G = (V, E, \mathbf{X})$: graph with nodes V, edges E, & feature matrix $\mathbf{X}$
2. $\mathbf{x}_i$: feature vector of node i
3. $\mathbf{h}_i$: hidden representation of node i
4. $\mathcal{N}_i$: set of neighboring nodes of node i
5. ψ: message function
6. ϕ: update function
7. $\bigoplus$: permutation invariant aggregation operator
8. $\mathbf{W}^{(l)}$: learnable weight matrix at layer l
9. d_i: degree of node i
10. α_{ij}: attention coefficient from node j to node i
11. e_{ij}: unnormalized attention score
12. $\mathbf{a}$: learnable attention vector
13. K: number of attention heads
14. $\epsilon^{(l)}$: learnable/fixed parameter in Graph Isomorphism Networks (GINs) at layer l
15. $\mathcal{R}$: set of relationship types in relationship GNNs
16. $\mathcal{N}_i^r$: set of neighbors of node i connected through relationship type r
17. $\mathbf{W}_r^{(l)}$: weight matrix specific to relationship type r at layer l

Reinforcement Learning & Imitation Learning notation

1. $\mathcal{S}$: set of states
2. $\mathcal{A}$: set of actions
3. P: transition probability function $P(s'|s, a)$ is prob. of $s \xrightarrow{a} s'$
4. R: reward function: $R(s, a, s')$ is reward for $s \xrightarrow{a} s'$
5. γ: discount factor
6. s, s_t, s_k: state (at a general, specific time step t, or decision k)
7. a, a_t, a_k: action (at a general, specific time step t, or decision k)
8. s': next state (resulting from state s & action a)
9. π: policy (mapping from states to actions or action probabilities)
10. $J(\pi)$: objective function; expected cumulative discounted reward for policy π
11. F: Sojourn time distribution in SMDP: $F(s, a, s', \tau)$ is prob. transition $s \xrightarrow{a} s'$ takes time τ
12. τ: time duration of a state transition in an SMDP
13. t_k: time at which kth decision is made (in SMDP, for discounting variable duration steps)
14. $V^\pi(s)$: state-value function: expected return from state s following policy π

15. $Q^\pi(s,a)$: action-value function: expected return from state s, taking action a, then following π
16. $A^\pi(s,a)$: $Q^\pi(s,a) - V^\pi(s)$
17. θ: learnable parameters of a function approximator (e.g., for a policy)
18. $\pi_\theta(a|s)$: policy parameterized by θ; outputs probability of taking action a in state s
19. $\nabla_\theta J(\pi_\theta)$: gradient of objective J w.r.t. policy parameters θ
20. d^π: on-policy state distribution under policy π
21. G_t: return: sum of discounted rewards from time step t onwards
22. N: number of data points (e.g., episod es in REINFORCE, expert state-action pairs in BC)
23. B: batch size for gradient updates (e.g., in A2C, PPO)
24. $L^{\text{CLIP}}(\theta)$: clipped surrogate objective function in PPO
25. $r_t(\theta)$: probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ in PPO, comparing current policy to old policy $\pi_{\theta_{\text{old}}}$
26. ϵ: hyperparameter in PPO defining clipping range $(1-\epsilon, 1+\epsilon)$
27. $\mathcal{D}_E$: dataset of expert demonstrations, typically pairs of (s_i, a_i^*)
28. a^*, a_i^*: action taken by an expert (in state s_i)
29. θ^*: optimal parameters for π_θ learned by minimizing behavior cloning closs
30. $L(\pi_\theta(s), a^*)$: loss function in Behavior Cloning, measures dissimilarity between $\pi_\theta(s)$ & a^*
31. $\mathcal{A}(s_k)$: set of available (candidate) actions in state s_k
32. y_{ij}: true label (e.g., 1 if operation O_{ij} is an expert action, 0 otherwise) for BC in JSSP
33. $\hat{y}_{ij}$: predicted probability by π_θ that operation O_{ij} is an expert action (BC for JSSP)

- 1. Introduction. This project addresses Job Shop Scheduling Problem (JSSP), 1 of most classical & essential problems in scheduling. Scheduling is a subset of combinatorial optimization, a branch of applied mathematics & CS that involves finding best solution from a finite set of possibilities. This set of feasible solutions is usually too large to perform an exhaustive search.

  – Dự án này giải quyết Bài toán Lập lịch Job Shop (JSSP), 1 trong những bài toán cổ điển & thiết yếu nhất trong lập lịch. Lập lịch là 1 tập hợp con của tối ưu hóa tổ hợp, 1 nhánh của toán học ứng dụng & khoa học máy tính (CS) liên quan đến việc tìm ra giải pháp tốt nhất từ 1 tập hợp hữu hạn các khả năng. Tập hợp các giải pháp khả thi này thường quá lớn để thực hiện tìm kiếm toàn diện.

  JSSP has special relevance in manufacturing sector (Gupta & Sivakumar, 2006; Wang et al., 2021; Zhang et al., 2023), where order in which tasks or operations are performed on each machine can significantly impact productivity. It also has important applications in hospitals (Pham & Klinkert, 2008; Sarfaraj et al., 2021) & supply chain management (Liao & Lin, 2019; Lei Cai & He, 2023).

  – JSSP có tầm quan trọng đặc biệt trong lĩnh vực sản xuất (Gupta & Sivakumar, 2006; Wang & cộng sự, 2021; Zhang & cộng sự, 2023), nơi thứ tự thực hiện các tác vụ hoặc thao tác trên mỗi máy có thể ảnh hưởng đáng kể đến năng suất. Nó cũng có những ứng dụng quan trọng trong bệnh viện (Pham & Klinkert, 2008; Sarfaraj & cộng sự, 2021) & quản lý chuỗi cung ứng (Liao & Lin, 2019; Lei Cai & He, 2023).

  Example 5. *A custom furniture workshop with 3 workers receives daily orders to manufacture various pieces. Today, they have been requested to build a chair, a table, & a cabinet. Each object represents a job. To complete them, they must undergo a series of phases, each requiring a different machine & worker. In particular, a cutting machine is 1st used to obtain wooden pieces needed to assemble each object. Then, chair & table must be sanded by a sanding machine before being assembled at assembly station. Cabinet requires sanding after assembly. Goal: complete all 3 jobs in shortest possible time, considering: each machine can only process 1 operation at a time. Can visualize times & order requirements in* Table 1.1: Example of a JSSP. Operations must be completed in order presented in table.

  – 1 xưởng sản xuất đồ nội thất theo yêu cầu với 3 công nhân nhận được đơn đặt hàng hàng ngày để sản xuất các sản phẩm khác nhau. Hôm nay, họ được yêu cầu đóng 1 chiếc ghế, 1 chiếc bàn & 1 tủ. Mỗi đồ vật đại diện cho 1 công việc. Để hoàn thành chúng, họ phải trải qua 1 loạt các giai đoạn, mỗi giai đoạn yêu cầu 1 máy móc & công nhân khác nhau. Cụ thể, trước tiên, máy cắt được sử dụng để lấy các mảnh gỗ cần thiết để lắp ráp từng đồ vật. Sau đó, ghế & bàn phải được chà nhám bằng máy chà nhám trước khi được lắp ráp tại trạm lắp ráp. Tủ cần được chà nhám sau khi lắp ráp. Mục tiêu: hoàn thành cả 3 công việc trong thời gian ngắn nhất có thể, lưu ý rằng: mỗi máy chỉ có thể xử lý 1 thao tác tại 1 thời điểm. Có thể hình dung thời gian & yêu cầu đặt hàng trong Bảng 1.1: Ví dụ về JSSP. Các thao tác phải được hoàn thành theo thứ tự được trình bày trong bảng.

  Note carefully considering scheduling of each task could lead to significant delays. E.g., if decide to schedule operations according to Fig. 1.1: Gantt chart representing a possible solution to Ex 1 (Table 1) prioritizing operations with shorter duration. On vertical axis are 3 machines involved: cutting, sanding, & assembly, while horizontal axis shows total processing time measured in hours. Each colored block represents an operation of a specific job on a particular machine, with its corresponding duration., we

would need a total of 30 hours of work, while with optimal solution, same result can be achieved in just 10 hours Fig. 1.2: Gantt chart representing an optimal solution to Ex 1.

– Lưu ý rằng việc cân nhắc kỹ lưỡng việc lập lịch trình cho từng tác vụ có thể dẫn đến sự chậm trễ đáng kể. Ví dụ: nếu quyết định lập lịch trình các hoạt động theo Hình 1.1: Biểu đồ Gantt thể hiện 1 giải pháp khả thi cho Ví dụ 1 (Bảng 1) bằng cách ưu tiên các hoạt động có thời gian thực hiện ngắn hơn. Trên trục tung có 3 máy: cắt, chà nhám, & lắp ráp, trong khi trục hoành thể hiện tổng thời gian xử lý được tính bằng giờ. Mỗi khối màu đại diện cho 1 hoạt động của 1 công việc cụ thể trên 1 máy cụ thể, với thời gian thực hiện tương ứng, chúng ta sẽ cần tổng cộng 30 giờ làm việc, trong khi với giải pháp tối ưu, kết quả tương tự có thể đạt được chỉ trong 10 giờ Hình 1.2: Biểu đồ Gantt thể hiện 1 giải pháp tối ưu cho Ví dụ 1.

Even in this simple example, achieve a productivity improvement of $> 20\%$. This difference can mean millions of dollars in economic savings for many manufacturing companies, especially in large-scale systems.

– Ngay cả trong ví dụ đơn giản này, năng suất cũng có thể cải thiện hơn 20%. Sự khác biệt này có thể mang lại hàng triệu đô la tiết kiệm kinh tế cho nhiều công ty sản xuất, đặc biệt là trong các hệ thống quy mô lớn.

Despite apparent simplicity of JSSP, it belongs to category of NP-hard problems (Garey et al., 1976), i.e., extremely unlikely that an efficient algorithm exists to solve problem. Finding an optimal solution in scenarios with hundreds or thousands of jobs & machines is virtually impossible.

– Mặc dù JSSP có vẻ đơn giản, nó thuộc loại bài toán NP-khó (Garey & cộng sự, 1976), tức là rất khó có thể có 1 thuật toán hiệu quả để giải quyết bài toán. Việc tìm ra giải pháp tối ưu trong các tình huống với hàng trăm hoặc hàng nghìn công việc & máy móc là gần như không thể.

Given this complexity, heuristics are frequently used to find solutions that, while not optimal, are quick to obtain & meet problem requirements at a level considered acceptable in practice (Boussaïd et al., 2013). Heuristic-based approaches are valuable when speed & practicality take priority over absolute precision. On other hand, "approximation algorithms" gives guarantees on how close found solution is to the optimization.

– Do tính phức tạp này, các phương pháp heuristic thường được sử dụng để tìm ra các giải pháp, mặc dù không tối ưu, nhưng có thể đạt được nhanh chóng & đáp ứng các yêu cầu của bài toán ở mức độ được coi là chấp nhận được trong thực tế (Boussaïd & cộng sự, 2013). Các phương pháp tiếp cận dựa trên heuristic có giá trị khi tốc độ & tính thực tiễn được ưu tiên hơn độ chính xác tuyệt đối. Mặt khác, "thuật toán xấp xỉ" đảm bảo độ gần đúng của giải pháp tìm được với phương án tối ưu hóa.

Another problem is great uncertainty in real situations, e.g. large factories. Sometimes, unexpected jobs arrive, some machines may temporarily break down, or duration of each task may not be deterministic, among other difficulties. These considerations quickly render any schedule obsolete, forcing operations to be rescheduled in real-life. Refer to this problem variant as a dynamic JSSP.

– 1 vấn đề khác là tính bất định lớn trong các tình huống thực tế, ví dụ như các nhà máy lớn. Đôi khi, các công việc bất ngờ xuất hiện, 1 số máy móc có thể tạm thời bị hỏng, hoặc thời gian thực hiện mỗi tác vụ có thể không xác định, cùng nhiều khó khăn khác. Những cân nhắc này nhanh chóng khiến bất kỳ lịch trình nào trở nên lỗi thời, buộc các hoạt động phải được lên lịch lại trong thực tế. Biến thể bài toán này được gọi là JSSP động.

Priority Dispatching Rules (PDRs) are frequently used to counter this uncertainty (Haupt, 1989). Instead of building complete schedule directly, they create it sequentially, determining which operation should be scheduled next. They can also be used as a starting point for methods that iteratively adjust a given schedule through local search, e.g. simulated annealing or tabu search. These rules are usually based on simple heuristics. E.g., 1st schedule was created using shortest processing time rule, prioritizing operations with shortest duration. Additionally, custom dispatching rules are often used. To effectively design these heuristics in real environments, knowledge & intuition of experts who understand most common cases & patterns in factory are often required (Kathawala & Allen, 1993).

– Quy tắc Điều phối Ưu tiên (PDR) thường được sử dụng để khắc phục sự bất định này (Haupt, 1989). Thay vì xây dựng lịch trình hoàn chỉnh trực tiếp, chúng tạo lịch trình theo trình tự, xác định thao tác nào nên được lên lịch tiếp theo. Chúng cũng có thể được sử dụng làm điểm khởi đầu cho các phương pháp điều chỉnh lặp lại 1 lịch trình nhất định thông qua tìm kiếm cục bộ, ví dụ: mô phỏng ủ hoặc tìm kiếm tabu. Các quy tắc này thường dựa trên các phương pháp heuristic đơn giản. Ví dụ: lịch trình thứ nhất được tạo bằng quy tắc thời gian xử lý ngắn nhất, ưu tiên các thao tác có thời gian thực hiện ngắn nhất. Ngoài ra, các quy tắc điều phối tùy chỉnh thường được sử dụng. Để thiết kế hiệu quả các phương pháp heuristic này trong môi trường thực tế, thường cần có kiến thức & trực giác của các chuyên gia hiểu rõ các trường hợp & mẫu phổ biến nhất trong nhà máy (Kathawala & Allen, 1993).

This complexity represents a great opportunity to automate heuristics of dispatching rules through ML techniques (Bengio et al., 2021). With ML, dispatcher learns directly from experience, improving its decisions over time by interacting with environment (Zhang et al., 2020; Park et al., 2021a; Ho et al., 2023; Yuan et al., 2023) or by imitating optimal solutions obtained through more computationally expensive algorithms (Lee & Kim, 2022, 2024). 1st family of methods leverages RL, a framework in which an agent interacts with an environment to maximize a reward. 2nd approach employs Imitation Learning (IL), a method for learning directly from optimal schedules through supervised learning.

– Sự phức tạp này là 1 cơ hội tuyệt vời để tự động hóa các phương pháp tìm kiếm quy tắc điều phối thông qua các kỹ thuật ML (Bengio & cộng sự, 2021). Với ML, bộ điều phối học hỏi trực tiếp từ kinh nghiệm, cải thiện các quyết định của mình theo thời gian bằng cách tương tác với môi trường (Zhang & cộng sự, 2020; Park & cộng sự, 2021a; Ho & cộng sự, 2023; Yuan & cộng sự, 2023) hoặc bằng cách mô phỏng các giải pháp tối ưu thu được thông qua các thuật toán tốn kém hơn về mặt tính toán (Lee & Kim, 2022, 2024). Nhóm phương pháp thứ nhất tận dụng RL (Học tập mô phỏng), 1 khuôn khổ trong đó 1 tác

nhân tương tác với môi trường để tối đa hóa phần thưởng. Cách tiếp cận thứ 2 sử dụng Học tập mô phỏng (IL), 1 phương pháp học trực tiếp từ các lịch trình tối ưu thông qua học có giám sát.

Moreover, despite its relevance & being a widely studied problem (Xiong et al., 2022), JSSP has received less attention regarding application of ML methods (Yuan et al., 2023) than other combinatorial optimization problems e.g. vehicle routing or TSP.

– Hơn nữa, mặc dù có liên quan & là 1 vấn đề được nghiên cứu rộng rãi (Xiong & cộng sự, 2022), JSSP ít được quan tâm liên quan đến việc áp dụng các phương pháp ML (Yuan & cộng sự, 2023) so với các vấn đề tối ưu hóa tổ hợp khác, ví dụ như định tuyến phương tiện hoặc TSP.

1 advantage of ML: it allows exploiting particular patterns of each use case. This property is especially relevant because, in most real scenarios, one is not interested in solving problem in general terms but rather a subset of problems with common characteristics (Bengio et al., 2021). Take case of custom furniture manufacturing workshop seen earlier. There may be different orders daily, but jobs encountered present similar patterns. E.g., all furniture will always require a cutting machine initially. A ML-based algorithm could be capable of specializing in solving JSSP instances (particular definitions of JSSP problem) related to this workshop efficiently.

– 1. Ưu điểm của ML: nó cho phép khai thác các mẫu cụ thể của từng trường hợp sử dụng. Đặc tính này đặc biệt quan trọng vì, trong hầu hết các tình huống thực tế, người ta không quan tâm đến việc giải quyết vấn đề 1 cách chung chung mà là 1 tập hợp con các vấn đề có đặc điểm chung (Bengio & cộng sự, 2021). Hãy lấy trường hợp xưởng sản xuất đồ nội thất theo yêu cầu đã đề cập ở trên làm ví dụ. Có thể có nhiều đơn hàng khác nhau mỗi ngày, nhưng các công việc gặp phải đều có các mẫu tương tự. E.g., tất cả đồ nội thất ban đầu luôn cần máy cắt. 1 thuật toán dựa trên ML có thể có khả năng chuyên biệt hóa để giải quyết các trường hợp JSSP (các định nghĩa cụ thể của bài toán JSSP) liên quan đến xưởng này 1 cách hiệu quả.

However, applying this learning-based approach also presents several challenges. 1 of most important is adequately representing relationships between tasks & resources & constraints that define order in which operations must be completed. Indices of machines & jobs can be permuted arbitrarily without changing underlying problem instance. Therefore, representations should be invariant to these permutations.

– Tuy nhiên, việc áp dụng phương pháp tiếp cận dựa trên học tập này cũng đặt ra 1 số thách thức. 1 trong những thách thức quan trọng nhất là biểu diễn đầy đủ mối quan hệ giữa các tác vụ & tài nguyên & các ràng buộc xác định thứ tự các thao tác phải được hoàn thành. Chỉ số của máy & công việc có thể được hoán vị tùy ý mà không làm thay đổi trường hợp bài toán cơ bản. Do đó, biểu diễn phải bất biến với các hoán vị này.

An effective way to address this complexity is through graphs. They represent elements (nodes) & connections between them (edges). E.g., a disjunctive graph represents operations as nodes, while dependencies between them or shared resources, e.g. machines, are modeled as edges Błazewicz et al., 2000). Graphs are a common way of representing combinatorial optimization problems. TSP, e.g., is about finding shortest path that traverses all nodes.

– 1 cách hiệu quả để giải quyết sự phức tạp này là thông qua đồ thị. Chúng biểu diễn các phần tử (nút) & & các kết nối giữa chúng (cạnh). E.g., 1 đồ thị phân biệt biểu diễn các thao tác dưới dạng các nút, trong khi các mối quan hệ phụ thuộc giữa chúng hoặc các tài nguyên dùng chung, ví dụ như máy móc, được mô hình hóa dưới dạng các cạnh (Błazewicz & cộng sự, 2000). Đồ thị là 1 cách phổ biến để biểu diễn các bài toán tối ưu hóa tổ hợp. E.g., TSP là về việc tìm đường đi ngắn nhất đi qua tất cả các nút.

GNNs (Scarselli et al., 2009; Gilmer et al., 2017; Cappart et al., 2021) are a powerful tool for solving tasks on graphs. Specifically, they can learn vector representations of graph nodes by iteratively aggregating features of neighboring nodes. An initial node feature could be its processing time, e.g. Once computed, vectors can be used to make predictions.

– Mạng nơ-ron nhân tạo (GNN) (Scarselli & cộng sự, 2009; Gilmer & cộng sự, 2017; Cappart & cộng sự, 2021) là 1 công cụ mạnh mẽ để giải quyết các bài toán trên đồ thị. Cụ thể, chúng có thể học biểu diễn vectơ của các nút đồ thị bằng cách tổng hợp lặp lại các đặc điểm của các nút lân cận. 1 đặc điểm ban đầu của nút có thể là thời gian xử lý của nút đó, ví dụ: Sau khi tính toán, các vectơ có thể được sử dụng để đưa ra dự đoán.

- 1.1. Contribution. Despite growing interest in learning PDRs with GNNs (Smit et al., 2025), there is still a lack of tools for efficient experimentation. Most of available implementations force users to experiment with a limited set of options.

  – Mặc dù sự quan tâm ngày càng tăng đối với việc học PDR bằng GNN (Smit & cộng sự, 2025), vẫn còn thiếu các công cụ để thử nghiệm hiệu quả. Hầu hết các triển khai hiện có đều buộc người dùng phải thử nghiệm với 1 tập hợp các tùy chọn hạn chế.

  As explored in Chap. 5, previous works differ in several design choices related to following open questions:

  * **What is optimal graph representation of a partial JSSP solution?** Cliques [A subset of nodes in a graph where every node is connected to every other one. In this case, cliques are formed between operations that share resources.] present in traditional disjunctive graphs can be expensive, computationally speaking, & lead to problems e.g. oversmoothing [A phenomenon in GNNs where node embeddings become overly similar, regardless of their original differences.]. Other alternatives e.g. Resource-Task graphs, create artificial machine nodes to remove such cliques. However, more experimentation is still needed to determine what optimal graph configuration for representing JSSP is.
    – **Biểu diễn đồ thị tối ưu của giải pháp JSSP 1 phần là gì?** Cliques [1 tập hợp con các nút trong đồ thị mà mọi nút đều được kết nối với mọi nút khác. Trong trường hợp này, các clique được hình thành giữa các thao tác chia sẻ tài nguyên.] có trong đồ thị phân ly truyền thống có thể tốn kém về mặt tính toán, & dẫn đến các vấn đề như làm mịn quá

mức [1 hiện tượng trong GNN mà các nhúng nút trở nên quá giống nhau, bất kể sự khác biệt ban đầu của chúng.]. Các phương án thay thế khác, ví dụ như đồ thị Tài nguyên-Nhiệm vụ, tạo ra các nút máy nhân tạo để loại bỏ các clique như vậy. Tuy nhiên, vẫn cần nhiều thử nghiệm hơn để xác định cấu hình đồ thị tối ưu để biểu diễn JSSP.

* **What is optimal initial set of node features (e.g., each operation)?** While some features can be very straightforward, e.g. including operation's processing time as part of this set, other features may require more creativity to be devised. Experiments run in this project suggest that these features can considerably impact performance. In particular, by only using `JobShopLib`'s defined features (i.e., not graph connectivity), we managed to outperform several works in literature.
  – **Tập hợp các đặc trưng nút ban đầu tối ưu (ví dụ: mỗi thao tác) là gì?** Trong khi 1 số đặc trưng có thể rất đơn giản, ví dụ: bao gồm thời gian xử lý của thao tác như 1 phần của tập hợp này, các đặc trưng khác có thể đòi hỏi sự sáng tạo hơn để thiết kế. Các thử nghiệm được thực hiện trong dự án này cho thấy những đặc trưng này có thể tác động đáng kể đến hiệu suất. Cụ thể, bằng cách chỉ sử dụng các đặc trưng được xác định của `JobShopLib` (tức là không sử dụng kết nối đồ thị), chúng tôi đã vượt trội hơn 1 số công trình trong tài liệu.
* **What operations should be considered available for dispatch?** Sometimes, restricting dispatcher from selecting probably suboptimal actions can lead to an increase in performance.
  – **Những hoạt động nào nên được coi là khả dụng để điều phối?** Đôi khi, việc hạn chế bộ điều phối chọn những hành động có thể không tối ưu có thể giúp tăng hiệu suất.
* **If using RL, what is best reward function?** Naive reward function gives a zero reward for every step, except for last one, & yields makespan in last step. However, this reward has an important downside – it is sparse. A sparse reward is one that occurs infrequently during an agent's learning process. This sparsity makes it difficult for agents to correlate actions with eventual rewards (credit assignment problem).
  – **Nếu sử dụng RL, hàm thưởng nào là tốt nhất?** Hàm thưởng ngây thơ cho phần thưởng bằng 0 cho mọi bước, ngoại trừ bước cuối cùng, & tạo ra makespan ở bước cuối cùng. Tuy nhiên, phần thưởng này có 1 nhược điểm quan trọng – nó thưa thớt. Phần thưởng thưa thớt là phần thưởng xuất hiện không thường xuyên trong quá trình học của tác nhân. Sự thưa thớt này khiến các tác nhân khó liên hệ hành động với phần thưởng cuối cùng (vấn đề phân bổ tín dụng).

Main contribution of this work is development of JobShopLib – a versatile Python library – & 2 highly flexible RL environments built upon it. This project also demonstrates these tools' capabilities through experiments requiring minimal setup.

– Đóng góp chính của công trình này là phát triển JobShopLib – 1 thư viện Python đa năng – & 2 môi trường RL cực kỳ linh hoạt được xây dựng dựa trên nó. Dự án này cũng chứng minh khả năng của các công cụ này thông qua các thử nghiệm chỉ yêu cầu thiết lập tối thiểu.

1 of trained models outperformed various GNN-based dispatchers while considering only operation features (no graph). Moreover, our GNN-based model has achieved very close to state-of-art results in large-scale problems.

– 1 trong số các mô hình được đào tạo đã vượt trội hơn nhiều bộ điều phối dựa trên GNN khác nhau khi chỉ xem xét các tính năng vận hành (không có biểu đồ). Hơn nữa, mô hình dựa trên GNN của chúng tôi đã đạt được kết quả rất gần với công nghệ tiên tiến trong các bài toán quy mô lớn.

* 1.1.1. JobShopLib: A Foundation for JSSP Research. `JobShopLib` is an open-source library that supports exploring all of these open questions. Key contributions of library include:
  · Basic data structures to create & manipulate JSSP instances & its solutions. These classes make creating, representing, & working with job shop scheduling problems easy.
  · Access to a dataset containing various benchmarks without downloading external files. `JobShopLib` also introduces a mechanism for storing instances & solutions that supports arbitrary metadata.
  · A random instance generation module for creating random instances with customizable sizes & properties.
  · Support for multiple solvers, including dispatching rules & an exact solver based on constraint programming capable of obtaining optimal schedules for small problems.
  · Support for arbitrary graph representations of JSSP, including specific built-in functions for building disjunctive & Resource-Task graphs.
  · A visualization module for plotting Gantt charts & graphs. It also supports creating videos or GIFs from sequences of dispatching decisions or from a dispatching rule solver.
  · 2 RL environments. The 1st one `SingleJobShopGraphEnv` allow users to solve a single JSSP instance. `MultiJobShopGraphEnv` builds upon this environment but supports training ML models over a distribution of instances. In particular, after solving 1 instance, a new random one is generated. Since this 2nd environment can be considered a wrapper of the 1st, refer in this project to `ingleJobShopGraphEnv` as `JobShopLib`'s environment. These 2 environments utilize almost all features described & they are main contribution of this project.

  An important consideration: `JobShopLib`'s environment is agnostic to learning method employed. In particular, it follows standard Gymnasium interface (Towers et al., 2024). RL algorithms are typically designed to interact with this interface. Moreover, despite not using reinforcement learning in our experiments, prove: RL environment generates training data valuable for imitation learning.
  – `JobShopLib` là 1 thư viện mã nguồn mở hỗ trợ việc khám phá tất cả những câu hỏi mở này. Các đóng góp chính của thư viện bao gồm:

· Cấu trúc dữ liệu cơ bản để tạo & thao tác các thể hiện JSSP & các giải pháp của nó. Các lớp này giúp việc tạo, biểu diễn, & làm việc với các bài toán lập lịch của Job Shop trở nên dễ dàng.

· Truy cập vào tập dữ liệu chứa nhiều điểm chuẩn khác nhau mà không cần tải xuống các tệp bên ngoài. `JobShopLib` cũng giới thiệu 1 cơ chế lưu trữ các thể hiện & các giải pháp hỗ trợ siêu dữ liệu tùy ý.

· 1 mô-đun tạo thể hiện ngẫu nhiên để tạo các thể hiện ngẫu nhiên với kích thước tùy chỉnh & các thuộc tính.

· Hỗ trợ nhiều bộ giải, bao gồm các quy tắc phân phối & 1 bộ giải chính xác dựa trên lập trình ràng buộc có khả năng thu được các lịch trình tối ưu cho các bài toán nhỏ.

· Hỗ trợ các biểu diễn đồ thị tùy ý của JSSP, bao gồm các hàm tích hợp cụ thể để xây dựng đồ thị & Tài nguyên-Nhiệm vụ rời rạc.

· 1 mô-đun trực quan hóa để vẽ biểu đồ Gantt & đồ thị. Nó cũng hỗ trợ tạo video hoặc GIF từ chuỗi các quyết định phân phối hoặc từ trình giải quy tắc phân phối.

· 2 môi trường RL. Môi trường đầu tiên `SingleJobShopGraphEnv` cho phép người dùng giải quyết 1 cá thể JSSP duy nhất. `MultiJobShopGraphEnv` được xây dựng dựa trên môi trường này nhưng hỗ trợ huấn luyện các mô hình ML trên 1 phân phối các cá thể. Cụ thể, sau khi giải quyết 1 cá thể, 1 cá thể ngẫu nhiên mới sẽ được tạo ra. Vì môi trường thứ 2 này có thể được coi là 1 lớp bao bọc của môi trường thứ 1, trong dự án này, hãy tham khảo `ingleJobShopGraphEnv` là môi trường của `JobShopLib`. Hai môi trường này sử dụng hầu hết tất cả các tính năng được mô tả & chúng là những đóng góp chính của dự án này.

1 điểm cần cân nhắc quan trọng: Môi trường của `JobShopLib` không phụ thuộc vào phương pháp học được sử dụng. Cụ thể, nó tuân theo giao diện Gymnasium tiêu chuẩn (Towers & cộng sự, 2024). Các thuật toán RL thường được thiết kế để tương tác với giao diện này. Hơn nữa, mặc dù không sử dụng học tăng cường trong các thí nghiệm của mình, hãy chứng minh: Môi trường RL tạo ra dữ liệu đào tạo có giá trị cho việc học mô phỏng.

∗ 1.1.2. Training of a GNN-based Dispatcher. Demonstrated practical capabilities of `JobShopLib`'s environment by showcasing its use in training a GNN-based dispatcher. This training uses imitation learning; GNN is trained to predict operations that lead to optimal schedules that a constraint programming solver previously found. I.e., GNN dispatcher learns to imitate scheduling decisions of an optimal solver. These optimal schedules can only be computed for small instances. Hope: model will generalize patterns found in small instances to bigger ones, where applying a constraint propramming solver would be computationally expensive.

– Huấn luyện Bộ điều phối dựa trên GNN. Đã chứng minh khả năng thực tế của môi trường `JobShopLib` bằng cách trình bày việc sử dụng nó trong việc đào tạo 1 bộ điều phối dựa trên GNN. Khóa đào tạo này sử dụng phương pháp học mô phỏng; GNN được đào tạo để dự đoán các hoạt động dẫn đến các lịch trình tối ưu mà 1 bộ giải lập trình ràng buộc đã tìm thấy trước đó. Tức là, bộ điều phối GNN học cách mô phỏng các quyết định lập lịch của 1 bộ giải tối ưu. Các lịch trình tối ưu này chỉ có thể được tính toán cho các trường hợp nhỏ. Hy vọng: mô hình sẽ khái quát hóa các mẫu được tìm thấy trong các trường hợp nhỏ thành các trường hợp lớn hơn, trong khi việc áp dụng 1 bộ giải lập trình ràng buộc sẽ tốn kém về mặt tính toán.

On top of this GNN-based dispatcher, trained a simpler baseline model to assess `JobShopLib`'s built-in features' expressiveness. This model does not use message passing mechanism of GNNs. Removing this component is equivalent to disconnecting graph & considering nodes in isolation. Show: these features are enough to outperform several state-of-art GNN-based approaches.

– Trên bộ điều phối dựa trên GNN này, chúng tôi đã huấn luyện 1 mô hình cơ sở đơn giản hơn để đánh giá khả năng biểu đạt của các tính năng tích hợp của `JobShopLib`. Mô hình này không sử dụng cơ chế truyền thông điệp của GNN. Việc loại bỏ thành phần này tương đương với việc ngắt kết nối đồ thị & xem xét các nút 1 cách riêng biệt. Hiển thị: các tính năng này đủ để vượt trội hơn 1 số phương pháp tiếp cận dựa trên GNN hiện đại.

Code of these experiments is available in https://github.com/Pabloo22/gnn_scheduler.

○ 1.2. Project's Structure. This project is divided into 2 parts:

∗ Part I discusses theoretical background needed to understand this project's contribution. It comprises Chaps. 2–5:

· Chap. 2 introduces formally JSSP & some core concepts that will be revised later, e.g. different types of schedules & solving methods. Moreover, show how to use basic features of `JobShopLib`, e.g. creating & solving JSSP instances, by showing boxes with brief code snippets & explanations.

· Chap. 3 explains how GNNs work, including main architectures employed by GNNs-based dispatchers. In particular, message passing framework is discussed & how it can be adapted to handle heterogeneous graphs (needed for JSSP).

· Chap. 4 introduces 2 main learning methods (RL & IL) for training DL-based dispatchers. It also introduces concept of Markov Decision Processes, theoretical framework used to model RL environments.

· Once theoretical background is set, in Chap. 5, analyze literature related to solve JSSP with a GNN-based dispatcher. Here, identify core design choices that researchers face. `JobShopLib` abstracts these features to allow users to customize & experiment with a wide range of options.

– Phần I thảo luận về nền tảng lý thuyết cần thiết để hiểu được đóng góp của dự án này. Phần này bao gồm các Chương 2–5:

· Chương 2 giới thiệu chính thức về JSSP & 1 số khái niệm cốt lõi sẽ được xem xét lại sau, ví dụ: các loại lịch biểu khác nhau & các phương pháp giải quyết. Hơn nữa, hướng dẫn cách sử dụng các tính năng cơ bản của `JobShopLib`, ví dụ: tạo & giải quyết các thể hiện JSSP, bằng cách hiển thị các hộp với các đoạn mã ngắn & giải thích.

· Chương 3 giải thích cách thức hoạt động của GNN, bao gồm các kiến trúc chính được sử dụng bởi các bộ điều phối dựa trên GNN. Đặc biệt, khung truyền thông điệp được thảo luận & cách nó có thể được điều chỉnh để xử lý các đồ thị không đồng nhất (cần thiết cho JSSP).

· Chương 4 giới thiệu 2 phương pháp học chính (RL & IL) để huấn luyện các bộ điều phối dựa trên DL. Nó cũng giới thiệu khái niệm về Quy trình Quyết định Markov, 1 khuôn khổ lý thuyết được sử dụng để mô hình hóa các môi trường RL.

· Sau khi nền tảng lý thuyết được thiết lập, trong Chương 5. Phân tích tài liệu liên quan đến việc giải quyết JSSP bằng bộ điều phối dựa trên GNN. Ở đây, hãy xác định các lựa chọn thiết kế cốt lõi mà các nhà nghiên cứu phải đối mặt. `JobShopLib` tóm tắt các tính năng này để cho phép người dùng tùy chỉnh & thử nghiệm với nhiều tùy chọn.

∗ Part II explains core contributions of this project. It consists of Chaps. 6–8:

· Chap. 6 introduces `JobShopLib`'s components necessary to solve JSSP with PDRs following a 1st-principles approach. These components include some of basic data structures & objects defined, as well as some of design patterns employed to maintain extensibility.

· Chap. 7 discusses RL environment's design & their components. It builds on concepts introduced in prev chaps & mentions built-in classes & functions that replicate some of specific design choices analyzed in Chap. 5.

· Chap. 8 contains experiments that prove environment's usefulness. 1st, evaluate quality of `JobShopLib`'s built-in node features. To accomplish this, show how a fully equipped GNN can be trained using our library & compare results with state of art.

· Chap. 9 reflects on ethical, environmental, & social aspects affecting our project.

– Phần II giải thích những đóng góp cốt lõi của dự án này. Dự án bao gồm các Chương 6-8:

· Chương 6 giới thiệu các thành phần cần thiết của `JobShopLib` để giải quyết JSSP với PDR theo phương pháp tiếp cận nguyên lý thứ nhất. Các thành phần này bao gồm 1 số cấu trúc dữ liệu cơ bản & các đối tượng đã được định nghĩa, cũng như 1 số mẫu thiết kế được sử dụng để duy trì khả năng mở rộng.

· Chương 7 thảo luận về thiết kế của môi trường RL & các thành phần của chúng. Phần này được xây dựng dựa trên các khái niệm đã giới thiệu trong các chương trước & đề cập đến các lớp & các hàm dựng sẵn sao chép 1 số lựa chọn thiết kế cụ thể đã được phân tích trong Chương 5.

· Chương 8 bao gồm các thử nghiệm chứng minh tính hữu ích của môi trường. Đầu tiên, hãy đánh giá chất lượng các tính năng nút dựng sẵn của `JobShopLib`. Để thực hiện điều này, hãy trình bày cách 1 GNN được trang bị đầy đủ có thể được huấn luyện bằng thư viện của chúng tôi & so sánh kết quả với công nghệ hiện đại.

· Chương 7–9 phản ánh các khía cạnh đạo đức, môi trường & xã hội ảnh hưởng đến dự án của chúng tôi.

Conclude project by evaluating project's successes & failures & suggesting future work to take project further.

– Kết thúc dự án bằng cách đánh giá thành công & thất bại của dự án & đề xuất công việc trong tương lai để đưa dự án tiến xa hơn.

I. BACKGROUND.

• 2. Job Shop Scheduling Problem. Job shop scheduling problem was formally formulated in 1950s(Johnson, 1954). Since then, it has been a relevant research topic due to its complexity & broad applicability in manufacturing, including semiconductors, automotive, & electronics production. Although there are numerous variations, in this project, address classical version:

◦ We have a job shop environment with a set of jobs $\mathcal{J}$ & machines $\mathcal{M}$. Size of a problem is denoted as $|\mathcal{J}| \times |\mathcal{M}|$.

◦ Each job $J_i \in \mathcal{J}$ consists of a sequence of operations $O_{i1} \to O_{i2} \to \cdots \to O_{in_i}$. These operations must be processed in a given order. Set of all operations in a problem is denoted as $\mathcal{O}$.

◦ Each operation is assigned to a machine $M_j \in \mathcal{M}$, with a processing time $p_{ij} \in \mathbb{N}$. Said machine can only process 1 operation at a time.

◦ A schedule is defined by assigning to each operations O_{ij} a start time $S_{ij} \in \mathbb{N}$ that satisfies mentioned constraints.

◦ Main objective: find a schedule that minimizes maximum completion time of all jobs, also known as makespan. This objective is formally defined as maximum completion time among all tasks $C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$.

– Bài toán lập lịch xưởng gia công được chính thức xây dựng vào những năm 1950 (Johnson, 1954). Kể từ đó, nó đã trở thành 1 chủ đề nghiên cứu quan trọng do tính phức tạp & khả năng ứng dụng rộng rãi trong sản xuất, bao gồm sản xuất chất bán dẫn, ô tô, & điện tử. Mặc dù có nhiều biến thể, nhưng trong dự án này, chúng tôi sẽ đề cập đến phiên bản cổ điển:

◦ Chúng ta có 1 môi trường xưởng gia công với 1 tập hợp các công việc $\mathcal{J}$ & máy móc $\mathcal{M}$. Kích thước của 1 bài toán được ký hiệu là $|\mathcal{J}| \times |\mathcal{M}|$.

◦ Mỗi công việc $J_i \in \mathcal{J}$ bao gồm 1 chuỗi các thao tác $O_{i1} \to O_{i2} \to \cdots \to O_{in_i}$. Các thao tác này phải được xử lý theo 1 thứ tự nhất định. Tập hợp tất cả các thao tác trong 1 bài toán được ký hiệu là $\mathcal{O}$.

◦ Mỗi thao tác được gán cho 1 máy $M_j \in \mathcal{M}$, với thời gian xử lý $p_{ij} \in \mathbb{N}$. Máy này chỉ có thể xử lý 1 thao tác tại 1 thời điểm.

◦ 1 lịch trình được định nghĩa bằng cách gán cho mỗi thao tác O_{ij} 1 thời điểm bắt đầu $S_{ij} \in \mathbb{N}$ thỏa mãn các ràng buộc đã đề cập.

- Mục tiêu chính: tìm 1 lịch trình tối thiểu hóa thời gian hoàn thành tối đa của tất cả các công việc, còn được gọi là makespan. Mục tiêu này được định nghĩa chính thức là thời gian hoàn thành tối đa giữa tất cả các tác vụ $C_{\max} = \max_{i,j}\{C_{ij} = S_{ij}+p_{ij}\}$.

In Sect. 2.2, show how problem can be visualized with a graph, which can be useful for understanding notation. In this work, focus on problems of this type: processing times are constant, & there are no setup times, due dates, or release dates, among other factors.

– Trong Phần 2.2, hãy trình bày cách biểu diễn bài toán bằng đồ thị, 1 phương pháp hữu ích để hiểu ký hiệu. Trong bài viết này, tập trung vào các bài toán thuộc loại này: thời gian xử lý là hằng số, & không có thời gian thiết lập, ngày đến hạn, hay ngày phát hành, cùng nhiều yếu tố khác.

Representing an Instance in JobShopLib. To define problem in `JobShopLib`, `JobShopInstance` is used. This class comprises a list of lists of `Operation` objects, which act as containers for properties of each operation. See how instance from Ex1 was defined:

```
from job_shop_lib import JobShopInstance, Operation

# each machine is represented with an ID (starting from 0)
CUTTING_MACHINE_ID = 0
SANDING_MACHINE_ID = 1
ASSEMBLY_MACHINE_ID = 2
table = [
Operation(CUTTING_MACHINE_ID, duration = 2),
Operation(SANDING_MACHINE_ID, duration = 2),
Operation(ASSEMBLY_STATION_ID, duration = 2),
]
chair = [
Operation(CUTTING_MACHINE_ID, duration = 1),
Operation(SANDING_MACHINE_ID, duration = 1),
Operation(ASSEMBLY_STATION_ID, duration = 1),
]
cabinet = [
Operation(CUTTING_MACHINE_ID, duration = 2),
Operation(ASSEMBLY_STATION_ID, duration = 3),
Operation(SANDING_MACHINE_ID, duration = 3),
]
jobs = [table, chair, cabinet]
instance = JobShopInstance(jobs)
```

Through this class, one can access statistics & properties of problem. Some examples are number of jobs, machines, or total duration per job or machine. See https://job-shop-lib.readthedocs.io/en/latest/api/job_shop_lib.html#job_shop_lib.JobShopInstance

- 2.1. Representing Solutions. A schedule or solution to problem Y is represented as a matrix of size $|\mathcal{M}| \times |\mathcal{J}|$ in which each row i represents sequence of operations that machine $M_i \in \mathcal{M}$ will process. E.g., optimal solution to Ex1 is a schedule that can be represented by matrix

$$Y = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

In this matrix, 1st row corresponds to cutting machine M_1, 2nd to sanding machine M_2, & 3rd to assembly station M_3. Numbers within matrix represent jobs that each machine will process in order. Jobs are indexed as J_1 for table, J_2 for chair, & J_3 for cabinet. E.g., 1st number in 1st row indicates: cutting machine M_1 will begin with cabinet J_3, then continue with table J_1, & finally with chair J_2. This same pattern applies to other machines, according to index shown in each row. Note: indices chosen to represent each machine or job are arbitrary, & any permutation of these would represent same problem.

– Biểu diễn Giải pháp. 1 lịch trình hoặc giải pháp cho bài toán Y được biểu diễn dưới dạng ma trận có kích thước $|\mathcal{M}| \times |\mathcal{J}|$, trong đó mỗi hàng i biểu diễn chuỗi các thao tác mà máy $M_i \in \mathcal{M}$ sẽ xử lý. Ví dụ: giải pháp tối ưu cho Ex1 là 1 lịch trình có thể được biểu diễn bằng ma trận

$$Y = \begin{pmatrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

Trong ma trận này, hàng thứ nhất tương ứng với máy cắt M_1, hàng thứ 2 tương ứng với máy chà nhám M_2, hàng thứ ba tương ứng với trạm lắp ráp M_3. Các số trong ma trận biểu diễn các công việc mà mỗi máy sẽ xử lý theo thứ tự. Các công việc được đánh số là J_1 cho bàn, J_2 cho ghế, & J_3 cho tủ. Ví dụ: số thứ nhất trong hàng thứ nhất cho biết: máy cắt M_1 sẽ bắt đầu với tủ J_3, sau đó tiếp tục với bàn J_1, & cuối cùng là ghế J_2. Quy luật tương tự này cũng áp dụng cho các máy khác, tùy theo chỉ số được hiển thị trong mỗi hàng. Lưu ý: các chỉ số được chọn để biểu thị cho mỗi máy hoặc công việc là tùy ý, & bất kỳ sự hoán vị nào của chúng đều sẽ biểu thị cùng 1 vấn đề.

∗ 2.1.1. Types of Schedules. To obtain a complete schedule with specific timings from a sequence matrix like Y, typically generate corresponding semi-active schedule. A semi-active schedule is constructed by starting each operation as early as possible, ensuring that no precedence constraints within a job are violated, required machine is available, & crucially, operation order specified by matrix Y for that machine is maintained. E.g., if matrix stipulates that operation O_{22} must be executed before O_{33} on their shared machine, O_{33} will begin only once both its preceding operation within same job e.g. O_{32} & its machine predecessor O_{22} have finished processing. Following this "start as early as possible" rule for a given sequence naturally eliminates unnecessary idle time for that specific sequence & guarantees finding schedule with smallest possible makespan among all schedules respecting matrix's sequence constraints. Considering semi-active schedules is generally sufficient, as set of all semi-active schedule is known to contain at least 1 optimal solution for regular objectives like makespan.

– Các loại Lịch trình. Để có được 1 lịch trình hoàn chỉnh với các thời gian cụ thể từ 1 ma trận trình tự như Y, thông thường hãy tạo ra 1 lịch trình bán chủ động tương ứng. 1 lịch trình bán chủ động được xây dựng bằng cách bắt đầu mỗi thao tác càng sớm càng tốt, đảm bảo rằng không có ràng buộc nào về thứ tự ưu tiên trong 1 công việc bị vi phạm, máy cần thiết phải khả dụng, & quan trọng là thứ tự thao tác do ma trận Y chỉ định cho máy đó được duy trì. Ví dụ: nếu ma trận quy định rằng thao tác O_{22} phải được thực thi trước O_{33} trên máy dùng chung của chúng, O_{33} sẽ chỉ bắt đầu khi cả thao tác trước đó trong cùng 1 công việc, ví dụ: O_{32} & máy tiền nhiệm của nó O_{22} đã hoàn tất xử lý. Tuân theo quy tắc "bắt đầu càng sớm càng tốt" này cho 1 trình tự nhất định sẽ loại bỏ thời gian nhàn rỗi không cần thiết cho trình tự cụ thể đó & đảm bảo tìm được lịch trình có khoảng thời gian chờ nhỏ nhất có thể trong số tất cả các lịch trình tuân theo các ràng buộc trình tự của ma trận. Việc xem xét các lịch trình bán chủ động thường là đủ, vì tập hợp tất cả các lịch trình bán chủ động đều chứa ít nhất 1 giải pháp tối ưu cho các mục tiêu thông thường như makespan.

While semi-active schedules are efficient for a given sequence, it may be possible to improve some of them further by changing the sequence itself (e.g. allowing an operation to "jump ahead" of another on the same machine if it doesn't delay anything else overall). Schedules where no such sequence-altering improvements (termed global left-shifts) are possible are called active schedules. This set is a subset of semi-active schedules & is theoretically important because it also forms a dominant set. I.e., we only need to search within active schedules to ensure we find an optimal solution.

– Mặc dù các lịch trình bán chủ động hiệu quả đối với 1 chuỗi nhất định, nhưng có thể cải thiện 1 số lịch trình hơn nữa bằng cách thay đổi chính chuỗi đó (ví dụ: cho phép 1 thao tác "nhảy lên trước" 1 thao tác khác trên cùng 1 máy nếu nó không làm chậm trễ bất kỳ thao tác nào khác). Các lịch trình không thể thực hiện các cải tiến thay đổi chuỗi như vậy (gọi là dịch chuyển trái toàn cục) được gọi là các lịch trình chủ động. Tập hợp này là 1 tập con của các lịch trình bán chủ động & về mặt lý thuyết rất quan trọng vì nó cũng tạo thành 1 tập trội. Tức là, chúng ta chỉ cần tìm kiếm trong các lịch trình chủ động để đảm bảo tìm thấy 1 giải pháp tối ưu.

Finally, an even smaller subset consists of non-delay schedules. These are active schedules with additional strict condition that no machine is ever kept idle if there is an operation ready & waiting for processing on that machine. Non-delay schedules are often computationally easier to generate & analyze, & frequently provide high-quality solutions. However, unlike active set, non-delay set is not guaranteed to contain an optimal schedule, as sometimes strategically inserting idle time can lead to a better overall makespan.

– Cuối cùng, 1 tập con thậm chí còn nhỏ hơn bao gồm các lịch trình không trễ. Đây là các lịch trình hoạt động với điều kiện nghiêm ngặt bổ sung là không bao giờ để máy nhàn rỗi nếu có 1 thao tác sẵn sàng & đang chờ xử lý trên máy đó. Các lịch trình không trễ thường dễ tính toán hơn & phân tích, & thường cung cấp các giải pháp chất lượng cao. Tuy nhiên, không giống như tập hợp hoạt động, tập hợp không trễ không được đảm bảo chứa 1 lịch trình tối ưu, vì đôi khi việc chèn thời gian nhàn rỗi 1 cách chiến lược có thể dẫn đến thời gian hoàn thành tổng thể tốt hơn.

Representing Schedules in JobShopLib. In `JobShopLib`, solutions are represented using `Schedule` class. This class comprises a list of lists of `ScheduledOperation` objects, which relate each operation to a machine & a start time. To define a schedule through solution matrix Y, can use static method `from_job_sequences` of `Schedule` class:

– **Trình diễn lịch biểu trong JobShopLib.** Trong `JobShopLib`, các giải pháp được biểu diễn bằng lớp `Schedule`. Lớp này bao gồm 1 danh sách các đối tượng danh sách `ScheduledOperation`, liên kết mỗi thao tác với 1 máy & thời gian bắt đầu. Để xác định 1 lịch trình giải quyết ma trận Y, có thể sử dụng phương thức tĩnh `from_job_sequences` của lớp `Schedule`:

```
# represent schedules in JobShopLib
from job_shop_lib import Schedule

## the id of each job is the index in the jobs list
TABLE_ID = 0
CHAIR_ID = 1
CABINET_ID = 2

cutting_machine_order = [CABINET_ID, TABLE_ID, CHAIR_ID]
sanding_machine_order = [TABLE_ID, CHAIR_ID, CABINET_ID]
assembly_station_order = [CABINET_ID, TABLE_ID, CHAIR_ID]
y = [cutting_machine_order, sanding_machine_order, assembly_station_order, ]
schedule = Schedule.from_job_sequences(instance, y)
```

∘ 2.2. Disjunctive Graph. To better visualize this problem, a disjunctive graph representation $G = (V, C \cup D)$ is commonly used (Błazewicz et al., 2000); this name is simply due to it being composed of 2 types of edges (C, D). V denotes set

of nodes, each representing an operation O_j. An initial node S & a final node T are also added. They serve as starting & ending points for all operations & can be considered operations with zero duration. Directed edges of 2 types connect nodes. Conjunctive edges C represent precedence relationships: if an operation O_{ij} must be completed before next operation $O_{i(j+1)}$, there is a directed arc from O_{ij} to $O_{i(j+1)}$. On other hand, disjunctive edges D are added between operations that require same machine M_j. These edges are not initially oriented, as they represent a resource conflict: machine can only process 1 operation at a time, so deciding which operation will be processed 1st will be necessary.

– Đồ thị phân ly. Để hình dung rõ hơn về vấn đề này, biểu diễn đồ thị phân ly $G = (V, C \cup D)$ thường được sử dụng (Błazewicz & cộng sự, 2000); tên này đơn giản là do nó bao gồm 2 loại cạnh (C, D). V biểu thị tập hợp các nút, mỗi nút biểu diễn 1 thao tác O_j. 1 nút đầu S & 1 nút cuối T cũng được thêm vào. Chúng đóng vai trò là điểm bắt đầu & điểm kết thúc cho tất cả các thao tác & có thể được coi là các thao tác có thời lượng bằng không. Các cạnh có hướng của 2 loại kết nối các nút. Các cạnh liên hợp C biểu diễn các mối quan hệ thứ tự ưu tiên: nếu 1 thao tác O_{ij} phải được hoàn thành trước thao tác tiếp theo $O_{i(j+1)}$, thì có 1 cung có hướng từ O_{ij} đến $O_{i(j+1)}$. Mặt khác, các cạnh rời rạc D được thêm vào giữa các thao tác yêu cầu cùng 1 máy M_j. Các cạnh này ban đầu không được định hướng, vì chúng biểu thị xung đột tài nguyên: máy chỉ có thể xử lý 1 thao tác tại 1 thời điểm, do đó cần phải quyết định thao tác nào sẽ được xử lý trước.

Disjunctive graphs can also represent solutions through appropriate orientation of disjunctive arcs Fig. 2.3: Disjunctive graph of Ex1 representing optimal solution presented in Fig. 1.2. Specifically, direction indicates order in which each machine will process each task. E.g., 1st node of 3rd row O_{31} is connected to 1st node of 1st row O_{11}. This node, in turn, points to node representing 1st task of 2nd row O_{21}. Therefore, cutting machine (represented by ID 0) will 1st process operation O_{31}, then O_{11}, & finally O_{21}.

– Đồ thị rời rạc cũng có thể biểu diễn các giải pháp thông qua hướng thích hợp của các cung rời rạc Hình 2.3: Đồ thị rời rạc của Ex1 biểu diễn giải pháp tối ưu được trình bày trong Hình 1.2. Cụ thể, hướng chỉ ra thứ tự mà mỗi máy sẽ xử lý từng tác vụ. Ví dụ: nút thứ nhất của hàng thứ 3 O_{31} được kết nối với nút thứ nhất của hàng thứ nhất O_{11}. Đến lượt mình, nút này trỏ đến nút biểu diễn tác vụ thứ nhất của hàng thứ 2 O_{21}. Do đó, máy cắt (được biểu diễn bằng ID 0) sẽ xử lý tác vụ O_{31} đầu tiên, sau đó là O_{11}, & cuối cùng là O_{21}.

- 2.3. Solving Problem. Various methods exist to solve JSSP. They can be classified as "exact" [KV18], heuristic, or metaheuristic (Boussaïd et al., 2013). Exact algorithms, e.g. Mixed Integer Linear Programming (MILP) or branch-&-bound, seek to find optimal solution but are limited to smaller problem instances due to their computational demands. Heuristic methods provide faster, albeit suboptimal, & solutions are often used in real-life applications where fast decision-making is crucial. This is category of priority dispatching rules. Finally, metaheuristic approaches, e.g. genetic algorithms, simulated annealing, & tabu search, balance solution quality & computational efficiency, making them suitable for larger & more complex problem instances. Their name comes from operating at a more general level ("meta") than traditional heuristics, meaning they can be applied to different combinatorial optimization problems.

  – Có nhiều phương pháp để giải JSSP. Chúng có thể được phân loại thành "chính xác" [KV18], heuristic hoặc metaheuristic (Boussaïd & cộng sự, 2013). Các thuật toán chính xác, ví dụ như Lập trình tuyến tính số nguyên hỗn hợp (MILP) hoặc branch-&-bound, tìm cách tìm ra giải pháp tối ưu nhưng bị giới hạn trong các trường hợp bài toán nhỏ hơn do yêu cầu tính toán của chúng. Các phương pháp heuristic cung cấp các giải pháp nhanh hơn, mặc dù không tối ưu, & thường được sử dụng trong các ứng dụng thực tế, nơi việc ra quyết định nhanh chóng là rất quan trọng. Đây là loại quy tắc phân bổ ưu tiên. Cuối cùng, các phương pháp metaheuristic, ví dụ như thuật toán di truyền, ủ mô phỏng, & tìm kiếm tabu, chất lượng giải pháp cân bằng & hiệu quả tính toán, khiến chúng phù hợp với các trường hợp bài toán lớn hơn & phức tạp hơn. Tên của chúng xuất phát từ việc hoạt động ở cấp độ tổng quát hơn ("meta") so với các heuristic truyền thống, nghĩa là chúng có thể được áp dụng cho các bài toán tối ưu hóa tổ hợp khác nhau.

  In this sect, describe Constraint Programming (CP), & exact method utilized in this work to generate a dataset of optimal instances which we trained an ML model. 2nd, go into detail about PDRs, a method used to compare performance of this model.

  – Trong phần này, mô tả Lập trình ràng buộc (CP), & phương pháp chính xác được sử dụng trong công trình này để tạo ra tập dữ liệu các trường hợp tối ưu mà chúng tôi đã đào tạo 1 mô hình ML. Thứ 2, đi sâu vào PDR, 1 phương pháp được sử dụng để so sánh hiệu suất của mô hình này.

  * 2.3.1. Constraint Programming. CP (Zhou, 1996; Rossi et al., 2006) is a paradigm for solving combinatorial search problems belonging to family of exact algorithms described above. This method allows us to find optimal solutions to small problems quickly. To solve problem, user must define a series of variables & constraints for a general optimizer to solve it. In our case, JSSP is encoded in following way:

    1. **Variables.**
       · S_{ij}: start time of operation O_{ij}.
       · C_{ij}: completion time of operation O_{ij}: $C_{ij} = S_{ij} + p_{ij}$.
    2. **Objective.** Minimize makespan $\max_{ij} C_{ij}$.
    3. **Constraints.**
       · Job sequencing constraints: completion time C_{ij} must be $\leq$ start time of next operation in job $S_{i(j+1)}$: $C_{ij} \leq S_{i(j+1)}$.
       · Machine non-overlapping constraints: for each machine $M_k \in \mathcal{M}$, none of operations O_{ij}, O_{pq} assigned to M_k can overlap in time: $S_{ij} \geq C_{pq}$ or $S_{pq} \geq C_{ij}$.

    – Lập trình Ràng buộc. CP (Zhou, 1996; Rossi & cộng sự, 2006) là 1 mô hình để giải quyết các bài toán tìm kiếm tổ hợp thuộc họ các thuật toán chính xác được mô tả ở trên. Phương pháp này cho phép chúng ta nhanh chóng tìm ra các giải

pháp tối ưu cho các bài toán nhỏ. Để giải quyết bài toán, người dùng phải định nghĩa 1 chuỗi các biến & ràng buộc để 1 trình tối ưu hóa tổng quát giải quyết. Trong trường hợp của chúng tôi, JSSP được mã hóa theo cách sau:

1. **Biến.**
   · S_{ij}: thời điểm bắt đầu của phép toán O_{ij}.
   · C_{ij}: thời điểm hoàn thành của phép toán O_{ij}: $C_{ij} = S_{ij} + p_{ij}$.
2. **Mục tiêu.** Giảm thiểu khoảng thời gian $\max_{ij} C_{ij}$.
3. **Ràng buộc.**
   · Ràng buộc về trình tự công việc: thời gian hoàn thành C_{ij} phải bằng ≤ thời gian bắt đầu của thao tác tiếp theo trong công việc $S_{i(j+1)}$: $C_{ij} \leq S_{i(j+1)}$.
   · Ràng buộc không chồng chéo máy: đối với mỗi máy $M_k \in \mathcal{M}$, không có thao tác nào trong số O_{ij}, O_{pq} được gán cho M_k có thể chồng chéo về mặt thời gian: $S_{ij} \geq C_{pq}$ hoặc $S_{pq} \geq C_{ij}$.

Once defined, use Google's CP-SAT optimizer from OR-Tools library (Perron & Didier, 2024). It searches for possible solutions that satisfy (SAT) constraints defined. It applies a broad range of techniques, each with strengths & weaknesses. Although each technique runs concurrently, once any of these processes finds a better solution, result is communicated between them so they can take advantage of this information (Krupke, 2024). In contrast, other exactalgorithms, such as those mentioned above, specialize in a single strategy that does not always turn out to be most efficient.

– Sau khi xác định, hãy sử dụng trình tối ưu hóa CP-SAT của Google từ thư viện OR-Tools (Perron & Didier, 2024). Trình tối ưu hóa này tìm kiếm các giải pháp khả thi thỏa mãn các ràng buộc (SAT) đã xác định. Trình tối ưu hóa này áp dụng 1 loạt các kỹ thuật, mỗi kỹ thuật đều có điểm mạnh & điểm yếu. Mặc dù mỗi kỹ thuật chạy đồng thời, nhưng khi bất kỳ quy trình nào trong số này tìm thấy giải pháp tốt hơn, kết quả sẽ được trao đổi giữa chúng để chúng có thể tận dụng thông tin này (Krupke, 2024). Ngược lại, các thuật toán chính xác khác, chẳng hạn như các thuật toán đã đề cập ở trên, tập trung vào 1 chiến lược duy nhất mà không phải lúc nào cũng hiệu quả nhất.

Using CP-SAT optimizer from OR-Tools in JobShopLib. `JobShopLib` provides `ORToolsSolver` class, which encodes problem in previously presented from transparently to user:

```
# use CP-SAT optimizer from OR-Tools in JobShopLib
from job_shop_lib.constraint_programming import ORToolsSolver

cp_sat_optimizer = ORToolsSolver()
optimal_schedule = cp_sat_optimizer(instance)
```

∗ 2.3.2. Priority Dispatching Rules. PDRs are 1 of most used methods in practice due to their simplicity & ability to be applied to dynamic environments. In a dynamic environment, new jobs may appear, or certain machines may temporarily break down, e.g. This uncertainty is something that other techniques, e.g. CP, struggle with. When using CP, redefining constraints & variables & solving problem again is necessary every time a change occurs in factory. This limitation, added to time needed to find an acceptable solution in large-scale problems, significantly hinders their application in real environments.

– Quy tắc điều phối ưu tiên. PDR là 1 trong những phương pháp được sử dụng nhiều nhất trong thực tế nhờ tính đơn giản & khả năng áp dụng trong môi trường động. Trong môi trường động, các công việc mới có thể xuất hiện, hoặc 1 số máy móc có thể tạm thời bị hỏng, ví dụ: . Sự không chắc chắn này là điều mà các kỹ thuật khác, ví dụ như CP, gặp khó khăn. Khi sử dụng CP, việc xác định lại các ràng buộc & biến & giải quyết lại vấn đề là cần thiết mỗi khi có thay đổi xảy ra trong nhà máy. Hạn chế này, cùng với thời gian cần thiết để tìm ra giải pháp chấp nhận được trong các bài toán quy mô lớn, cản trở đáng kể việc ứng dụng chúng trong môi trường thực tế.

Dispatching rules do not present these limitations because they create schedule sequentially. Specifically, each time a machine becomes idle [Classical definition of PDRs assumes that only operations that can start immediately are eligible to be dispatched. Use of alternative definitions is analyzed in Sect. 5.2 & Subsect. 6.4.1.], a heuristic that tends to be simple & easy to interpret computes priority of each operation that can be assigned to that machine (Holthaus & Rajendran, 1997). Table 2.1: Some of standard PDRs in JSSP shows classic PDRs for the previously defined JSSP scenario. Rule | Description

· Shortest Processing Time (SPT): Selects operation with shortest processing time p_{ij}

· Most Work Remaining (MWKR): Prioritizes job with greatest total remaining processing time

· Most Operations Remaining (MOR): Selects job with most pending operations

· First Come First Served (FCFS): Selects job with lowest ID.

· Random: Selects a random operation from those available.

– Quy tắc điều phối không gặp phải những hạn chế này vì chúng tạo lịch trình tuần tự. Cụ thể, mỗi khi máy nhàn rỗi [Định nghĩa cổ điển về PDR giả định rằng chỉ những tác vụ có thể bắt đầu ngay lập tức mới đủ điều kiện được điều phối. Việc sử dụng các định nghĩa thay thế được phân tích trong Mục 5.2 & Tiểu mục 6.4.1.], 1 phương pháp tìm kiếm có xu hướng đơn giản & dễ diễn giải sẽ tính toán mức độ ưu tiên của từng tác vụ có thể được gán cho máy đó (Holthaus & Rajendran, 1997). Bảng 2.1: 1 số PDR tiêu chuẩn trong JSSP hiển thị các PDR cổ điển cho kịch bản JSSP đã được định nghĩa trước đó. Quy tắc | Mô tả

· Thời gian xử lý ngắn nhất (SPT): Chọn thao tác có thời gian xử lý ngắn nhất p_{ij}

· Công việc còn lại nhiều nhất (MWKR): Ưu tiên công việc có tổng thời gian xử lý còn lại lớn nhất

· Hoạt động còn lại nhiều nhất (MOR): Chọn công việc có nhiều thao tác đang chờ xử lý nhất

· Ai đến trước được phục vụ trước (FCFS): Chọn công việc có ID thấp nhất.

· Ngẫu nhiên: Chọn 1 thao tác ngẫu nhiên từ các thao tác khả dụng.

An important observation: by defining time an operation is selected as "when a machine becomes idle", implicitly discard possibility of keeping machine waiting for a potentially more productive operation that is not yet available. Therefore, this definition also acts as a heuristic that limits space of available actions to operations that can begin immediately.
– 1 lưu ý quan trọng: bằng cách xác định thời điểm 1 thao tác được chọn là "khi máy trở nên nhàn rỗi", ngầm loại trừ khả năng máy phải chờ 1 thao tác có khả năng hiệu quả hơn nhưng chưa sẵn sàng. Do đó, định nghĩa này cũng hoạt động như 1 phương pháp tìm kiếm giới hạn không gian hành động khả dụng thành các thao tác có thể bắt đầu ngay lập tức.
Using PDRs in JobShopLib. `JobShopLib` provides `DispatchingRuleSolver` class, which is capable of applying any assignment rule that a callable (e.g., a function) can represent. Furthermore, library itself includes previously mentioned rules, meaning you do not need to program them. These can be invoked using a text string. E.g., solving previous instance using most work remaining rule can be achieved as follows:

```
# use PDRs in JobShopLib
from job_shop_lib.dispatching import DispatchRuleSolver

solver_mwkr = DispatchRuleSolver("most_work_remaining")
schedule_mwkr = solver_mwkr(instance)
```

- 3. GNNs. Combinatorial optimization presents unique challenges for traditional ML. 1 is input data often presents irregular structures with a dynamic number of elements & relationships between them. In JSSP, each instance can have a different number of jobs & machines, & precedence relationships between operations can vary significantly. Due to these characteristics, JSSP is naturally expressed as a graph. For this reason, graph neural networks are an efficient tool for processing the problem (Bengio et al., 2021).

  – Tối ưu hóa tổ hợp đặt ra những thách thức độc đáo cho ML truyền thống. 1. Dữ liệu đầu vào thường có cấu trúc bất quy tắc với số lượng phần tử động & mối quan hệ giữa chúng. Trong JSSP, mỗi trường hợp có thể có số lượng công việc & máy móc khác nhau, & mối quan hệ thứ tự ưu tiên giữa các thao tác có thể thay đổi đáng kể. Do những đặc điểm này, JSSP thường được biểu diễn dưới dạng đồ thị. Vì lý do này, mạng nơ-ron đồ thị là 1 công cụ hiệu quả để xử lý vấn đề (Bengio & cộng sự, 2021).

  1 of most important properties of these models is their permutation invariance. I.e., order in which graph nodes are presented does not affect model's final result. In context of JSSP, this is crucial since order in which machines or jobs are enumerated should not affect solution found. E.g., given 2 instances of same problem where only numbering order of machines changes, a traditional neural network might produce different results. At same time, a GNN guarantees same output. Possessing this property is essential because it means that, in practice, network will need less data during its training than a non-permutation-invariant counterpart.

  – 1 trong những đặc tính quan trọng nhất của các mô hình này là tính bất biến hoán vị. Nghĩa là, thứ tự các nút đồ thị được hiển thị không ảnh hưởng đến kết quả cuối cùng của mô hình. Trong bối cảnh JSSP, điều này rất quan trọng vì thứ tự liệt kê máy móc hoặc công việc không ảnh hưởng đến giải pháp được tìm thấy. E.g., với 2 trường hợp của cùng 1 bài toán mà chỉ có thứ tự đánh số máy móc thay đổi, mạng nơ-ron truyền thống có thể tạo ra các kết quả khác nhau. Đồng thời, mạng nơ-ron nhân tạo (GNN) đảm bảo đầu ra giống nhau. Việc sở hữu đặc tính này là rất cần thiết vì trên thực tế, mạng sẽ cần ít dữ liệu hơn trong quá trình huấn luyện so với mạng không bất biến hoán vị.

  Another fundamental advantage is their ability to process graphs of variable size. This property is essential for JSSP, as it allows us to train a model with small instances & apply it to instances with a larger number of machines & jobs. This generalization capability has been demonstrated empirically in several previous works (Zhang et al., 2020; Park et al., 2021a).

  – 1 lợi thế cơ bản khác là khả năng xử lý đồ thị có kích thước thay đổi. Tính chất này rất cần thiết cho JSSP, vì nó cho phép chúng ta huấn luyện 1 mô hình với các thể hiện nhỏ & áp dụng nó vào các thể hiện có số lượng máy móc & công việc lớn hơn. Khả năng khái quát hóa này đã được chứng minh bằng thực nghiệm trong 1 số công trình trước đây (Zhang & cộng sự, 2020; Park & cộng sự, 2021a).

  Furthermore, these models also facilitate incorporation of domain-specific knowledge. E.g., in JSSP, can encode relevant information in nodes, e.g. their processing time, number of remaining operations in their job, or even maximum deadline by which we want to complete operation. This flexibility allows us to represent various problem variants through these algorithms. Although not considered in this work, encoding information in edges is also possible.

  – Hơn nữa, các mô hình này cũng tạo điều kiện thuận lợi cho việc tích hợp kiến thức chuyên ngành. E.g., trong JSSP, có thể mã hóa thông tin liên quan trong các nút, chẳng hạn như thời gian xử lý, số thao tác còn lại trong tác vụ của chúng, hoặc thậm chí thời hạn tối đa mà chúng ta muốn hoàn thành thao tác. Tính linh hoạt này cho phép chúng ta biểu diễn nhiều biến thể bài toán khác nhau thông qua các thuật toán này. Mặc dù không được xem xét trong nghiên cứu này, việc mã hóa thông tin trong các cạnh cũng khả thi.

  - 3.1. Recap on Neural Networks. Before diving into GNNs, convenient to briefly revisit fundamentals of traditional Neural Networks (NNs), also known as ANNs. Inspired by biological neurons, these models are fundamental building block of modern DP architectures [LBH15].

– Tóm tắt về Mạng Nơ-ron. Trước khi đi sâu vào Mạng Nơ-ron Nhân tạo (GNN), chúng ta nên xem lại sơ lược những kiến thức cơ bản về Mạng Nơ-ron Nhân tạo (NN) truyền thống, còn được gọi là ANN. Lấy cảm hứng từ các nơ-ron sinh học, những mô hình này là nền tảng cơ bản của kiến trúc DP hiện đại [LBH15].

Core component of an NNs is artificial neuron. It receives a set of input signals, computes a weighted sum of these inputs, adds a bias term, & then applies a nonlinear activation function to produce an output. Mathematically, output z of a single neuron processing an input vector $\mathbf{x} \in \mathbb{R}^n$ can be expressed as

$$z = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) = \sigma(\mathbf{w}^\top \mathbf{x} + b).$$

Here $\mathbf{w} \in \mathbb{R}^n$ represents vector of weights, $b \in \mathbb{R}$: bias term, & $\sigma(\cdot)$ denotes activation function. Common choices for σ include sigmoid function, hyperbolic tangent tanh, or Rectified Linear Unit (ReLU) (Nair & Hinton, 2010). Introduction of nonlinearity via activation function is crucial, as it allows network to model complex, nonlinear relationships within data.

– Thành phần cốt lõi của mạng nơ-ron nhân tạo là nơ-ron nhân tạo. Nó nhận 1 tập hợp các tín hiệu đầu vào, tính tổng có trọng số của các tín hiệu đầu vào này, thêm 1 số hạng độ lệch, sau đó áp dụng 1 hàm kích hoạt phi tuyến tính để tạo ra đầu ra. Về mặt toán học, đầu ra z của 1 nơ-ron đơn lẻ xử lý 1 vectơ đầu vào $\mathbf{x} \in \mathbb{R}^n$ có thể được biểu thị như sau:

$$z = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) = \sigma(\mathbf{w}^\top \mathbf{x} + b).$$

Trong đó $\mathbf{w} \in \mathbb{R}^n$ biểu diễn vectơ trọng số, $b \in \mathbb{R}$: số hạng độ lệch, & $\sigma(\cdot)$ biểu thị hàm kích hoạt. Các lựa chọn phổ biến cho σ bao gồm hàm sigmoid, tan hyperbolic tanh, hoặc Đơn vị tuyến tính chỉnh lưu (ReLU) (Nair & Hinton, 2010). Việc đưa tính phi tuyến tính vào hàm kích hoạt là rất quan trọng, vì nó cho phép mạng mô hình hóa các mối quan hệ phi tuyến tính phức tạp trong dữ liệu.

Neurons are typically organized into layers to form a network. Most prevalent architecture is Multilayer Perception (MLP), which comprises an input layer, 1 or more hidden layers, & an output layer. In a standard MLP, neurons in 1 layer are fully connected to neurons in subsequent layer. Input layer receives raw feature vector, hidden layers process this information through successive transformations, learning increasingly abstract representations, & output layer generates final prediction, e.g. a classification score or a regression value. This computation can be formulated as:

$$\mathbf{h}^{(l+1)} = \sigma^{(l)}(W^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}),$$

where $W^{(l)}$ is weight matrix connecting layer l to layer $l+1$, $\mathbf{b}^{(l)}$: bias vector for layer l, $\mathbf{h}^{(0)} = \mathbf{x}$ represents initial input features, & $\sigma^{(l)}$: activation function applied elementwise to neurons in layer l.

– Các neuron thường được tổ chức thành nhiều lớp để tạo thành 1 mạng lưới. Kiến trúc phổ biến nhất là Nhận thức Đa lớp (MLP), bao gồm 1 lớp đầu vào, 1 hoặc nhiều lớp ẩn, & 1 lớp đầu ra. Trong 1 MLP tiêu chuẩn, các neuron ở lớp đầu vào được kết nối hoàn toàn với các neuron ở lớp tiếp theo. Lớp đầu vào nhận vectơ đặc trưng thô, các lớp ẩn xử lý thông tin này thông qua các phép biến đổi liên tiếp, học các biểu diễn ngày càng trừu tượng, & lớp đầu ra tạo ra dự đoán cuối cùng, ví dụ: điểm phân loại hoặc giá trị hồi quy. Phép tính này có thể được xây dựng như sau:

$$\mathbf{h}^{(l+1)} = \sigma^{(l)}(W^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}),$$

trong đó $W^{(l)}$ là ma trận trọng số kết nối lớp l với lớp $l+1$, $\mathbf{b}^{(l)}$: vectơ độ lệch cho lớp l, $\mathbf{h}^{(0)} = \mathbf{x}$ biểu diễn các đặc trưng đầu vào ban đầu, & $\sigma^{(l)}$: hàm kích hoạt được áp dụng từng phần tử cho các nơ-ron trong lớp l.

Training an NN involves optimizing its parameters (weights W & biases $\mathbf{b}$) to minimize a loss function L, which measures discrepancy between network's predictions & ground truth. This optimization is achieved using backpropagation algorithm (Rumelhart et al., 1986), which provides an efficient way to compute gradient of loss function w.r.t. network parameters by applying chain rule recursively. These gradients are then used by an optimization algorithm, e.g. Stochastic Gradient Descent (SGD) or its variants like Adam (Kingma & Ba, 2015), to iteratively update parameters in direction that minimizes loss.

– Việc huấn luyện 1 mạng nơ-ron nhân tạo bao gồm việc tối ưu hóa các tham số của nó (trọng số W & độ lệch $\mathbf{b}$) để giảm thiểu hàm mất mát L, hàm này đo lường sự khác biệt giữa dự đoán của mạng & giá trị thực tế. Việc tối ưu hóa này được thực hiện bằng thuật toán lan truyền ngược (Rumelhart & cộng sự, 1986), cung cấp 1 cách hiệu quả để tính toán độ dốc của hàm mất mát theo các tham số mạng bằng cách áp dụng quy tắc chuỗi 1 cách đệ quy. Các độ dốc này sau đó được sử dụng bởi 1 thuật toán tối ưu hóa, ví dụ như Stochastic Gradient Descent (SGD) hoặc các biến thể của nó như Adam (Kingma & Ba, 2015), để cập nhật các tham số theo hướng giảm thiểu mất mát.

NN function as powerful universal function approximators, capable of learning complex mappings between inputs & outputs (Hornik et al., 1989; Cybenko, 1989). However, conventional architectures like MLPs operate on fixed-size input vectors. This property makes them inherently ill-suited for processing data with irregular structures, variable sizes, or explicit relational dependencies, e.g. graphs representing JSSP instances. For this reason, GNNs were designed to operate on & leverage information present in graph-structured data.

– Mạng nơ-ron nhân tạo (NN) hoạt động như các bộ xấp xỉ hàm phổ quát mạnh mẽ, có khả năng học các phép ánh xạ phức tạp giữa đầu vào & đầu ra (Hornik & cộng sự, 1989; Cybenko, 1989). Tuy nhiên, các kiến trúc thông thường như MLP hoạt động trên các vectơ đầu vào có kích thước cố định. Đặc tính này khiến chúng vốn không phù hợp để xử lý dữ liệu có cấu trúc bất thường, kích thước thay đổi hoặc phụ thuộc quan hệ rõ ràng, ví dụ: đồ thị biểu diễn các thể hiện JSSP. Vì lý do này, GNN được thiết kế để hoạt động trên & tận dụng thông tin có trong dữ liệu có cấu trúc đồ thị.

◦ 3.2. Message Passing Algorithm. Message passing algorithm is a fundamental idea behind graph neural networks (Gilmer et al., 2017). At a high level, concept of "message passing" refers to transmission of information between connected nodes.

– Thuật toán Truyền tin. Thuật toán truyền tin là 1 ý tưởng cơ bản đằng sau mạng nơ-ron đồ thị (Gilmer & cộng sự, 2017). Ở cấp độ cao hơn, khái niệm "truyền tin" đề cập đến việc truyền thông tin giữa các nút được kết nối.

Formally, a graph, in context of GNNs, can be defined as $G = (V, E, X)$, where V represents set of nodes & E set of edges. Rows of matrix X represent feature vectors $\mathbf{x}_i$ for each node $i \in V$. Algorithm operates in 2 phases:

1. **Aggregation.** Each node collects information from its neighbors.
2. **Update.** Each node updates its representation using aggregated information.

In particular, for each node i, updated feature vectors for layer $l+1$ can be expressed as [NQBH: missing $\psi^{(l)}$ in original Bachelor thesis]

$$\mathbf{h}_i^{(l+1)} = \phi^{(l)}\left(\mathbf{h}_i^{(l)}, \bigoplus_{j \in \mathcal{N}_i} \psi^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)})\right),$$

where $\mathcal{N}_i$ denotes set of neighboring nodes of i, $\psi^{(l)}$ is a "message function" that returns a message $\mathbf{m}_j$ for each neighbor j, $\bigoplus$ is a permutation invariant aggregation operator, $\phi^{(l)}$ is an update function, & $\mathbf{h}_i^{(0)} = \mathbf{x}_i$. Most common options for aggregation operator $\bigoplus$ are:

1. Sum: $\bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_j = \sum_{j \in N(i)} \mathbf{m}_i$.
2. Maximum: $\bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_j = \max_{j \in N(i)} \mathbf{m}_i$.
3. Mean: $\bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_j = \frac{1}{|\mathcal{N}_i|} \sum_{j \in N(i)} \mathbf{m}_i$.

On other hand, update function $\phi^{(l)}$ is typically implemented as a neural network that combines current node representation with aggregated messages $\mathbf{m}$ from its neighbors. A common implementation is:

$$\phi(\mathbf{x}_i, \mathbf{m}) = \text{MLP}([\mathbf{x}_i, \mathbf{m}]),$$

where $[,]$ denotes vector concatenation.

– Về mặt hình thức, 1 đồ thị, trong bối cảnh của GNN, có thể được định nghĩa là $G = (V, E, X)$, trong đó V biểu diễn tập hợp các nút & E tập hợp các cạnh. Các hàng của ma trận X biểu diễn các vectơ đặc trưng $\mathbf{x}_i$ cho mỗi nút $i \in V$. Thuật toán hoạt động theo 2 giai đoạn:

1. **Tổng hợp.** Mỗi nút thu thập thông tin từ các nút lân cận.
2. **Cập nhật.** Mỗi nút cập nhật biểu diễn của mình bằng thông tin tổng hợp.

Cụ thể, đối với mỗi nút i, các vectơ đặc trưng được cập nhật cho lớp $l+1$ có thể được biểu thị dưới dạng [NQBH: thiếu $\psi^{(l)}$ trong luận văn Cử nhân gốc]

$$\mathbf{h}_i^{(l+1)} = \phi^{(l)}\left(\mathbf{h}_i^{(l)}, \bigoplus_{j \in \mathcal{N}_i} \psi^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)})\right),$$

trong đó $\mathcal{N}_i$ biểu thị tập hợp các nút lân cận của i, $\psi^{(l)}$ là 1 "hàm thông điệp" trả về 1 thông điệp $\mathbf{m}_j$ cho mỗi nút lân cận j, $\bigoplus$ là 1 Toán tử tổng hợp bất biến hoán vị, $\phi^{(l)}$ là 1 hàm cập nhật, & $\mathbf{h}_i^{(0)} = \mathbf{x}_i$. Các tùy chọn phổ biến nhất cho toán tử tổng hợp $\bigoplus$ là:

1. Tổng: $\bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_j = \sum_{j \in N(i)} \mathbf{m}_i$.
2. Tối đa: $\bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_j = \max_{j \in N(i)} \mathbf{m}_i$.
3. Trung bình: $\bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_j = \frac{1}{|\mathcal{N}_i|} \sum_{j \in N(i)} \mathbf{m}_i$.

Mặt khác, hàm cập nhật $\phi^{(l)}$ thường được triển khai dưới dạng mạng nơ-ron kết hợp biểu diễn nút hiện tại với các thông điệp tổng hợp $\mathbf{m}$ từ các nút lân cận. 1 triển khai phổ biến là:

$$\phi(\mathbf{x}_i, \mathbf{m}) = \text{MLP}([\mathbf{x}_i, \mathbf{m}]),$$

trong đó $[,]$ biểu thị phép nối vectơ.

By stacking multiple layers of this algorithm, each node can receive information from increasingly distant nodes in graph. Specifically, with L layers, each node can receive information from nodes up to L steps away. In context of JSSP, this property allows dispatching decisions to consider interactions between distant operations in graph.

– Bằng cách xếp chồng nhiều lớp của thuật toán này, mỗi nút có thể nhận thông tin từ các nút ở khoảng cách ngày càng xa trong đồ thị. Cụ thể, với L lớp, mỗi nút có thể nhận thông tin từ các nút cách xa tới L bước. Trong ngữ cảnh JSSP, thuộc tính này cho phép các quyết định phân phối xem xét tương tác giữa các thao tác ở xa trong đồ thị.

This framework can support various architectures, each with its advantages & disadvantages. In following sects, examine some of most relevant ones for JSSP.

– Khung này có thể hỗ trợ nhiều kiến trúc khác nhau, mỗi kiến trúc đều có ưu điểm & nhược điểm riêng. Trong các phần sau, hãy xem xét 1 số kiến trúc phù hợp nhất với JSSP.

* 3.2.1. Graph Convolutional Networks. Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) represent 1 of 1st & most influential architectures based on message-passing algorithm. Design is inspired by traditional convolutional neural networks (CNNs), extending convolution operation, widely used in image processing, to domain of graphs.
– Mạng Tích chập Đồ thị. Mạng Tích chập Đồ thị (GCN) (Kipf & Welling, 2017) đại diện cho 1 trong những kiến trúc đầu tiên & có ảnh hưởng nhất dựa trên thuật toán truyền thông điệp. Thiết kế được lấy cảm hứng từ mạng nơ-ron tích chập (CNN) truyền thống, mở rộng phép toán tích chập, được sử dụng rộng rãi trong xử lý ảnh, sang lĩnh vực đồ thị.
In a GCN, update of a node's representation can be expressed more specifically as

$$\mathbf{h}_j^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i\cup\{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(l)}\mathbf{h}_j^{(l)}\right),$$

where d_i, d_j are degrees of nodes i, j resp., $W^{(l)}$ is a learnable weight matrix, & σ: a nonlinear activation function (e.g., ReLU).
– Trong GCN, việc cập nhật biểu diễn của 1 nút có thể được biểu diễn cụ thể hơn như sau

$$\mathbf{h}_j^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i\cup\{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(l)}\mathbf{h}_j^{(l)}\right),$$

trong đó d_i, d_j là bậc của các nút i, j tương ứng, $W^{(l)}$ là ma trận trọng số có thể học được, & σ: 1 hàm kích hoạt phi tuyến tính (ví dụ: ReLU).
In a traditional CNN, a convolution applies a filter that averages neighboring pixels with fixed weights. Similarly, a GCN averages features of neighboring nodes, but normalizes according to each node's degree $\frac{1}{\sqrt{d_i d_j}}$. By taking into account degree of neighboring node u, importance of highly connected nodes is reduced. Otherwise, vectors of nodes with very high degrees could significantly contribute to homogenizing vectors of remaining nodes, especially after applying several layers of message passing. This phenomenon, known as "over-smoothing" is a well-known problem in GNNs (Rusch et al., 2023). In case of JSSP, e.g., it would mean that nodes representing each operation would have very similar vectors, which would make it impossible for model to distinguish which operation to dispatch. However, in practice, optimal normalization coefficient may vary depending on problem.
– Trong CNN truyền thống, phép tích chập áp dụng bộ lọc lấy trung bình các điểm ảnh lân cận với trọng số cố định. Tương tự, GCN lấy trung bình các đặc điểm của các nút lân cận, nhưng chuẩn hóa theo bậc $\frac{1}{\sqrt{d_i d_j}}$ của mỗi nút. Bằng cách tính đến bậc của nút lân cận u, tầm quan trọng của các nút có kết nối cao sẽ giảm đi. Mặt khác, các vectơ của các nút có bậc rất cao có thể góp phần đáng kể vào việc đồng nhất các vectơ của các nút còn lại, đặc biệt là sau khi áp dụng nhiều lớp truyền thông điệp. Hiện tượng này, được gọi là "làm mịn quá mức", là 1 vấn đề nổi tiếng trong GNN (Rusch & cộng sự, 2023). E.g., trong trường hợp JSSP, điều đó có nghĩa là các nút biểu diễn cho mỗi thao tác sẽ có các vectơ rất giống nhau, khiến mô hình không thể phân biệt được thao tác nào cần phân phối. Tuy nhiên, trên thực tế, hệ số chuẩn hóa tối ưu có thể thay đổi tùy thuộc vào vấn đề.
This formulation can be interpreted as a simplified version of message passing algorithm where aggregation function is a weighted average by node degree & update function is a linear transformation followed by a nonlinearity. However, GCNs have certain limitations: all neighboring nodes contribute in a fixed manner to update. This limitation motivated development of more sophisticated architectures e.g. graph attention networks.
– Công thức này có thể được hiểu là 1 phiên bản đơn giản hóa của thuật toán truyền tin nhắn, trong đó hàm tổng hợp là 1 giá trị trung bình có trọng số theo bậc nút & hàm cập nhật là 1 phép biến đổi tuyến tính theo sau là 1 phép phi tuyến tính. Tuy nhiên, GCN có 1 số hạn chế nhất định: tất cả các nút lân cận đều đóng góp vào quá trình cập nhật theo 1 cách cố định. Hạn chế này đã thúc đẩy sự phát triển của các kiến trúc phức tạp hơn, ví dụ như mạng lưới chú ý đồ thị.
* 3.2.2. Graph Attention Networks. Graph Attention Networks (GATs) (Veličković et al., 2018) extend capability of GCNs through incorporation of attention mechanisms. Similar to how GCNs extend concept of convolution in images to graphs, GATs generalize concept of self-attention used by transformers (Vaswani et al., 2017). In fact, transformer architecture can be considered a GAT specialized in fully connected graphs, where each token (subword in a sentence) is considered a node.
– Mạng Chú ý Đồ thị. Mạng Chú ý Đồ thị (GAT) (Veličković & cộng sự, 2018) mở rộng khả năng của GCN thông qua việc kết hợp các cơ chế chú ý. Tương tự như cách GCN mở rộng khái niệm tích chập trong hình ảnh sang đồ thị, GAT khái quát hóa khái niệm tự chú ý được sử dụng bởi các bộ biến đổi (Vaswani & cộng sự, 2017). Trên thực tế, kiến trúc bộ biến đổi có thể được coi là 1 GAT chuyên biệt trong các đồ thị kết nối đầy đủ, trong đó mỗi token (từ phụ trong câu) được coi là 1 nút.
Central idea: use an attention mechanism – a learned weighted average across a set of nodes (tokens in case of transformers). In particular, for each pair of connected nodes i, j, a learned attention coefficient α_{ij} determines importance of node j's information for updating node i's representation. This way, it solves limitation of GCNs, which use fixed coefficients determined solely by node's degree. Formally, update of a node in a GAT layer is expressed as

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i\cup\{i\}} \alpha_{ij}^{(l)} W^{(l)}\mathbf{h}_j^{(l)}\right).$$

Attention coefficients are calculated using an attention mechanism $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that calculates $e_{ij}^{(l)}$, unnormalized importance of node i for node j based on their features:

$$e_{ij}^{(l)} = a(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}).$$

– Ý tưởng chính: sử dụng cơ chế chú ý – 1 giá trị trung bình có trọng số đã học trên 1 tập hợp các nút (token trong trường hợp máy biến áp). Cụ thể, đối với mỗi cặp nút được kết nối i, j, 1 hệ số chú ý đã học α_{ij} xác định tầm quan trọng của thông tin của nút j trong việc cập nhật biểu diễn của nút i. Bằng cách này, nó giải quyết được hạn chế của GCN, vốn sử dụng các hệ số cố định được xác định hoàn toàn bởi bậc của nút. Về mặt hình thức, việc cập nhật 1 nút trong lớp GAT được biểu diễn như sau

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij}^{(l)} W^{(l)} \mathbf{h}_j^{(l)}\right).$$

Hệ số chú ý được tính toán bằng cơ chế chú ý $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ tính toán $e_{ij}^{(l)}$, tầm quan trọng chưa chuẩn hóa của nút i đối với nút j dựa trên các đặc trưng của chúng:

$$e_{ij}^{(l)} = a(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}).$$

Once these coefficients have been calculated for all neighbors $j \in \mathcal{N}_i$ of i, coefficients are normalized using softmax function:

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik}^{(l)})}.$$

Note this model is agnostic to attention mechanism used. E.g., in original paper, a single-layer neural network was used:

$$e_{ij} = \text{LeakyReLU}(\mathbf{a}^\top [W\mathbf{h}_j, W\mathbf{h}_i]),$$

where $[,]$ denotes concatenation operation, $\mathbf{a}$: a vector of learnable parameters, W: linear transformation matrix, & LeakyReLU is activation function:

$$\text{LeakyReLu}(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{if } x \leq 0, \end{cases}$$

where α: a small positive value (0.2 in this case) that allows a nonzero gradient for negative inputs. An importannt observation: in this approach, a linear transformation W is applied to feature vectors, folowed by a concatenation & another linear transformation $\mathbf{a}^\top$. Being consecutive linear operations, they can mathematically collapse into a single linear transformation. To understand why, examine equation $\mathbf{a}^\top [W\mathbf{h}_i, W\mathbf{h}_j]$ more closely. Despite involving a concatenation operation, this can actually be decomposed into a simple linear function of input node features. Specficially, if partititon attention vector $\mathbf{a}$ into 2 parts, $\mathbf{a} = [\mathbf{a}_1, \mathbf{a}_2]$, corresponding to dimensions used for $W\mathbf{h}_j$ & $W\mathbf{h}_i$ resp., then

$$\mathbf{a}^\top [W\mathbf{h}_j, W\mathbf{h}_i] = [\mathbf{a}_1^\top W\mathbf{h}_j, \mathbf{a}_2^\top W\mathbf{h}_i].$$

This can be further simplified by defining $W_1 = \mathbf{a}_1^\top W, W_2 = \mathbf{a}_2^\top W$ yielding

$$\mathbf{a}^\top [W\mathbf{h}_j, W\mathbf{h}_i] = [W_1\mathbf{h}_j, W_2\mathbf{h}_i].$$

Thus, without LeakyReLU, sequence of linear transformations followed by concatenation & another linear transformation would mathematically collapse into a single weighted sum of original node features. This limits model's ability to capture complex relationships between nodes. It is for this reason: it has recently been proposed to modify this attention mechanism as follows (Brody et al., 2022):

$$e_{ij} = \mathbf{a}^\top \text{LeakyReLU}([W\mathbf{h}_j, W\mathbf{h}_i]).$$

– Sau khi tính toán các hệ số này cho tất cả các lân cận $j \in \mathcal{N}_i$ của i, các hệ số được chuẩn hóa bằng hàm softmax:

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik}^{(l)})}.$$

Lưu ý rằng mô hình này không phụ thuộc vào cơ chế chú ý được sử dụng. E.g., trong bài báo gốc, mạng nơ-ron 1 lớp đã được sử dụng:

$$e_{ij} = \text{LeakyReLU}(\mathbf{a}^\top [W\mathbf{h}_j, W\mathbf{h}_i]),$$

trong đó $[,]$ biểu thị phép nối, $\mathbf{a}$: 1 vectơ tham số có thể học được, W: ma trận biến đổi tuyến tính, & LeakyReLU là hàm kích hoạt:

$$\text{LeakyReLu}(x) = \begin{cases} x & \text{nếu } x > 0, \\ \alpha x & \text{nếu } x \leq 0, \end{cases}$$

trong đó α: 1 giá trị dương nhỏ (trong trường hợp này là 0,2) cho phép gradient khác không đối với các đầu vào âm. 1 quan sát quan trọng: trong phương pháp này, 1 phép biến đổi tuyến tính W được áp dụng cho các vectơ đặc trưng, theo sau là 1 phép nối & 1 phép biến đổi tuyến tính khác $\mathbf{a}^\top$. Là các phép toán tuyến tính liên tiếp, về mặt toán học, chúng có thể được thu gọn thành 1 phép biến đổi tuyến tính duy nhất. Để hiểu lý do, hãy xem xét kỹ hơn phương trình $\mathbf{a}^\top[W\mathbf{h}_i, W\mathbf{h}_j]$. Mặc dù bao gồm 1 phép nối, phương trình này thực sự có thể được phân tích thành 1 hàm tuyến tính đơn giản của các đặc trưng nút đầu vào. Cụ thể, nếu phân chia vectơ chú ý $\mathbf{a}$ thành 2 phần, $\mathbf{a} = [\mathbf{a}_1, \mathbf{a}_2]$, tương ứng với các chiều được sử dụng cho $W\mathbf{h}_j$ & $W\mathbf{h}_i$ tương ứng, thì

$$\mathbf{a}^\top[W\mathbf{h}_j, W\mathbf{h}_i] = [\mathbf{a}_1^\top W\mathbf{h}_j, \mathbf{a}_2^\top W\mathbf{h}_i].$$

Điều này có thể được đơn giản hóa hơn nữa bằng cách định nghĩa $W_1 = \mathbf{a}_1^\top W, W_2 = \mathbf{a}_2^\top W$, tạo ra

$$\mathbf{a}^\top[W\mathbf{h}_j, W\mathbf{h}_i] = [W_1\mathbf{h}_j, W_2\mathbf{h}_i].$$

Do đó, nếu không có LeakyReLU, chuỗi các phép biến đổi tuyến tính theo sau bởi phép nối & 1 phép biến đổi tuyến tính khác về mặt toán học sẽ bị thu gọn thành 1 tổng trọng số duy nhất của các đặc trưng nút ban đầu. Điều này hạn chế khả năng của mô hình trong việc nắm bắt các mối quan hệ phức tạp giữa các nút. Vì lý do này: gần đây người ta đã đề xuất sửa đổi cơ chế chú ý này như sau (Brody & cộng sự, 2022):

$$e_{ij} = \mathbf{a}^\top \text{LeakyReLU}([W\mathbf{h}_j, W\mathbf{h}_i]).$$

By introducing LeakyReLU nonlinearity between linear transformations, model can learn more complex functions that better capture interactions between nodes. This seemingly simple change allows model to approximate any desired attention function thanks to universal approximation theorem (Hornik et al., 1989). For this reason, this latter variant is most used in practice. However, in general, any attention mechanism can be used.

– Bằng cách áp dụng tính phi tuyến tính LeakyReLU giữa các phép biến đổi tuyến tính, mô hình có thể học các hàm phức tạp hơn, nắm bắt tốt hơn các tương tác giữa các nút. Sự thay đổi tưởng chừng đơn giản này cho phép mô hình xấp xỉ bất kỳ hàm chú ý mong muốn nào nhờ định lý xấp xỉ phổ quát (Hornik & cộng sự, 1989). Vì lý do này, biến thể sau được sử dụng nhiều nhất trong thực tế. Tuy nhiên, nhìn chung, bất kỳ cơ chế chú ý nào cũng có thể được sử dụng.

On other hand, multiple attention "heads" are employed in parallel to improve stability & expressive power of model (in same way as transformers). To do this, aforementioned process is replicated independently K times using different parameters. This way, each head can learn to capture different types of relationships between nodes. Final aggregation is obtained by concatenating or averaging outputs of all heads:

1. Average: $\mathbf{h}_i^{(l+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i\cup\{i\}}\alpha_{ij}^{(l,k)}W^{(l,k)}\mathbf{h}_j^{(l)}\right)$

2. Concatenation: [...]

here denote concatenation $\|$.

– Mặt khác, nhiều "đầu" chú ý được sử dụng song song để cải thiện tính ổn định & khả năng biểu đạt của mô hình (tương tự như bộ biến đổi). Để làm được điều này, quá trình nói trên được lặp lại độc lập K lần bằng các tham số khác nhau. Bằng cách này, mỗi đầu có thể học cách nắm bắt các loại mối quan hệ khác nhau giữa các nút. Tổng hợp cuối cùng thu được bằng cách nối hoặc lấy trung bình các đầu ra của tất cả các đầu:

1. Trung bình: $\mathbf{h}_i^{(l+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i\cup\{i\}}\alpha_{ij}^{(l,k)}W^{(l,k)}\mathbf{h}_j^{(l)}\right)$.

2. Nối: [...]

ở đây ký hiệu nối $\|$.

In summary, a GAT layer with multi-head attention operates as follows: each neighboring node $j \in \mathcal{N}_i$ sends a vector of attention coefficients $\boldsymbol{\alpha}_{ij} = [\alpha_{ij}^{(1)}, \ldots, \alpha_{ij}^{(K)}]$, where each element corresponds to a different attention head. These coefficients are used to compute K independent linear combinations of the neighbors' features $\mathbf{h}_j$. Subsequently, these representations are aggregated (typically through concatenation or averaging) to obtain updated representation of node i, denoted as $\mathbf{h}_i^{(l+1)}$.

– Tóm lại, 1 lớp GAT với sự chú ý đa đầu hoạt động như sau: mỗi nút lân cận $j \in \mathcal{N}_i$ gửi 1 vectơ hệ số chú ý $\boldsymbol{\alpha}_{ij} = [\alpha_{ij}^{(1)}, \ldots, \alpha_{ij}^{(K)}]$, trong đó mỗi phần tử tương ứng với 1 đầu chú ý khác nhau. Các hệ số này được sử dụng để tính K tổ hợp tuyến tính độc lập của các đặc trưng $\mathbf{h}_j$ của các nút lân cận. Sau đó, các biểu diễn này được tổng hợp (thường thông qua phép nối hoặc lấy trung bình) để thu được biểu diễn cập nhật của nút i, được ký hiệu là $\mathbf{h}_i^{(l+1)}$.

∗ 3.2.3. Graph Isomorphism Networks. GINs emerge as an architecture designed to maximize discriminative power in graph processing (Xu et al., 2019). GIN bases its design on Weisfeiler-Lehman (WL) isomorphism test, an algorithm for determining whether 2 graphs are non-isomorphic. It aims to develop a GNN whose representational power is equivalent to WL test.

– Mạng Đồng cấu Đồ thị. GIN nổi lên như 1 kiến trúc được thiết kế để tối đa hóa khả năng phân biệt trong xử lý đồ thị (Xu & cộng sự, 2019). GIN dựa trên phép thử đồng cấu Weisfeiler-Lehman (WL), 1 thuật toán dùng để xác định xem 2 đồ thị có phải là không đồng cấu hay không. Mục tiêu của nó là phát triển 1 GNN có khả năng biểu diễn tương đương với phép thử WL.

Objective: ensure that each layer's aggregation & update functions can distinguish between different multisets of neighbor representations as well as WL test. Mathematically, this requires update function to be capable of learning an injective

mapping for these multisets. A function $f : X \to Y$ is injective if $\forall x_1, x_2 \in X$, $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$. This property, applied within layer update, is crucial for preserving structural information & distinguishing nodes with different neighborhood structures.

– Mục tiêu: đảm bảo rằng các hàm tổng hợp & cập nhật của mỗi lớp có thể phân biệt giữa các tập hợp đa biểu diễn lân cận khác nhau cũng như kiểm tra WL. Về mặt toán học, điều này đòi hỏi hàm cập nhật phải có khả năng học 1 ánh xạ đơn ánh cho các tập hợp đa này. 1 hàm $f : X \to Y$ là đơn ánh nếu $\forall x_1, x_2 \in X$, $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$. Thuộc tính này, được áp dụng trong cập nhật lớp, rất quan trọng để bảo toàn thông tin cấu trúc & phân biệt các nút có cấu trúc lân cận khác nhau.

Update of a node's representation in a GIN layer is expressed as

$$\mathbf{h}_i^{(l+1)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(l)} \right),$$

where $\epsilon^{(l)}$ is a learnable or fixed parameter that determines relative importance of target node vs. its neighbors.

– Việc cập nhật biểu diễn của 1 nút trong lớp GIN được biểu thị như sau

$$\mathbf{h}_i^{(l+1)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(l)} \right),$$

trong đó $\epsilon^{(l)}$ là 1 tham số có thể học được hoặc cố định, xác định tầm quan trọng tương đối của nút mục tiêu so với các nút lân cận.

GIN differs from previous architectures in 2 fundamental aspects:

1. **Sum aggregation.** Unlike weighted averaging or attention mechanisms used in GCN & GAT, GIN employs summation as its neighborhood aggregation function. Xu et al. (2019) proved: summation is most expressive aggregation function for multisets. It is capable of preserving maximum information about neighborhood feature distribution, mirroring counting aspect of WL test. Mean or max aggregators, in contrast, can map different multisets to same representation, limiting their discriminative power. E.g., mean & max aggregators cannot distinguish these 2 different sets of neighbor features: $\text{mean}(\{1,1,1\}) = \max(\{1,1,1\}) = 1, \text{mean}(\{1,1\}) = \max(\{1,1\}) = 1$.
2. **MLP for update.** Aggregated representation (sum of neighbors plus node's own representation) is processed by an MLP. As universal function approximators (Hornik et al., 1989), MLPs can learn arbitrarily complex functions. Importantly, they can approximate injective functions required to ensure: different input multisets (representing different local structures) are mapped to different output embeddings, thus maximizing layer's ability to distinguish nodes based on their neighborhood structure.

– GIN khác với các kiến trúc trước đây ở 2 khía cạnh cơ bản:

1. **Tổng hợp.** Không giống như các cơ chế trung bình có trọng số hoặc chú ý được sử dụng trong GCN & GAT, GIN sử dụng phép tổng làm hàm tổng hợp lân cận. Xu & cộng sự (2019) đã chứng minh: phép tổng hợp là hàm tổng hợp biểu diễn tốt nhất cho các tập hợp đa. Nó có khả năng bảo toàn tối đa thông tin về phân phối đặc trưng lân cận, phản ánh khía cạnh đếm của phép thử WL. Ngược lại, các bộ tổng hợp trung bình hoặc tối đa có thể ánh xạ các tập hợp đa khác nhau vào cùng 1 biểu diễn, hạn chế khả năng phân biệt của chúng. Ví dụ: các bộ tổng hợp trung bình & tối đa không thể phân biệt 2 tập hợp đặc trưng lân cận khác nhau này: $\text{mean}(\{1,1,1\}) = \max(\{1,1,1\}) = 1, \text{mean}(\{1,1\}) = \max(\{1,1\}) = 1$.
2. **MLP để cập nhật.** Biểu diễn tổng hợp (tổng của các lân cận cộng với biểu diễn riêng của nút) được xử lý bởi 1 MLP. Là các hàm xấp xỉ phổ quát (Hornik & cộng sự, 1989), MLP có thể học các hàm phức tạp tùy ý. Quan trọng hơn, chúng có thể xấp xỉ các hàm đơn ánh cần thiết để đảm bảo: các tập hợp đa đầu vào khác nhau (biểu diễn các cấu trúc cục bộ khác nhau) được ánh xạ thành các nhúng đầu ra khác nhau, do đó tối đa hóa khả năng phân biệt các nút dựa trên cấu trúc lân cận của chúng.

◦ 3.3. Relational Graph Neural Networks. Relational Graph Neural Networks (RGNNs) extend GNN paradigm to handle graphs with different types of relationships between nodes. This generalization is crucial for modeling complex systems where interactions between entities can be of a diverse nature. E.g., in context of JSSP, this means: network can process information coming through disjunctive edges (connecting operations that share a machine) & conjunctive edges (connecting operations of same job) differently. Similarly, this paradigm allows designing of new graph representations by introducing nodes of different types (e.g., machine nodes).

– Mạng Nơ-ron Đồ thị Quan hệ. Mạng Nơ-ron Đồ thị Quan hệ (RGNN) mở rộng mô hình GNN để xử lý đồ thị với các loại mối quan hệ khác nhau giữa các nút. Sự khái quát hóa này rất quan trọng để mô hình hóa các hệ thống phức tạp, nơi các tương tác giữa các thực thể có thể mang tính đa dạng. E.g., trong bối cảnh JSSP, điều này có nghĩa là: mạng có thể xử lý thông tin đến từ các cạnh rời rạc (kết nối các thao tác dùng chung 1 máy) & các cạnh liên hợp (kết nối các thao tác cùng 1 tác vụ) theo cách khác nhau. Tương tự, mô hình này cho phép thiết kế các biểu diễn đồ thị mới bằng cách đưa vào các nút thuộc các loại khác nhau (ví dụ: nút máy).

Fundamental idea behind these architectures: relationships are not simply binary links between nodes but possess their own semantics that must be considered during message aggregation. Thus, when a node updates its representation, it must take into account not only characteristics of its neighbors but also specific nature of each connection that links it to them. This approach allows network to learn patterns specific to each type of relationship, improving its ability to model systems with heterogeneous interactions.

– Ý tưởng cơ bản đằng sau những kiến trúc này: các mối quan hệ không chỉ đơn thuần là các liên kết nhị phân giữa các nút mà còn sở hữu ngữ nghĩa riêng cần được xem xét trong quá trình tổng hợp thông điệp. Do đó, khi 1 nút cập nhật biểu diễn của mình, nó không chỉ phải tính đến đặc điểm của các nút lân cận mà còn phải tính đến bản chất cụ thể của từng kết nối liên kết nó với chúng. Cách tiếp cận này cho phép mạng học các mẫu cụ thể cho từng loại mối quan hệ, cải thiện khả năng mô hình hóa các hệ thống có tương tác không đồng nhất.

Message aggregation in these networks must adapt to process multiple types of relationships simultaneously. For this, typically, different transformations are employed for different types of edges, allowing network to learn how each one should influence update of node features. I.e., a different weight matrix is applied for each type of edge.

– Việc tổng hợp thông điệp trong các mạng này phải thích ứng để xử lý đồng thời nhiều loại mối quan hệ. Để làm được điều này, thông thường, các phép biến đổi khác nhau được sử dụng cho các loại cạnh khác nhau, cho phép mạng học cách mỗi phép biến đổi ảnh hưởng đến việc cập nhật các đặc điểm của nút. Tức là, 1 ma trận trọng số khác nhau được áp dụng cho mỗi loại cạnh.

E.g., with GCNs, can modify update equation to account for different types of relationships (Schlichtkrull et al., 2018). Let $\mathcal{R}$ be set of relationship types & $\mathcal{N}_i^r$ set of neighbors of node i connected through a relationship of type r. Update equation can be expressed as:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,j,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right),$$

where $W_r^{(l)}$ is a weight matrix specific to each relationship type r & $c_{i,j,r}$ is a normalization constant that can be learned or fixed beforehand, e.g., $|\mathcal{N}_i^r|$.

– E.g., với GCN, có thể sửa đổi phương trình cập nhật để tính đến các loại mối quan hệ khác nhau (Schlichtkrull & cộng sự, 2018). Giả sử $\mathcal{R}$ là tập hợp các kiểu quan hệ & $\mathcal{N}_i^r$ là tập hợp các lân cận của nút i được kết nối thông qua 1 mối quan hệ kiểu r. Phương trình cập nhật có thể được biểu thị như sau:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,j,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right),$$

trong đó $W_r^{(l)}$ là ma trận trọng số cụ thể cho từng loại quan hệ r & $c_{i,j,r}$ là hằng số chuẩn hóa có thể được học hoặc cố định trước, ví dụ: $|\mathcal{N}_i^r|$.

∗ 3.3.1. Relational Message Passing. Similarly, relational variants of the other presented architectures can be defined. In general, we can extend the message passing framework

$$\mathbf{h}_i^{(l+1)} = \phi\left(\mathbf{h}_i^{(l)}, \bigoplus_{j\in\mathcal{N}_i}\psi^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)})\right)$$

to handle heterogeneous graphs containing multiple node types $\mathcal{T}_V$ & edge types $r = (t_{\text{src}}, \text{rel}, t_{\text{dst}}) \in \mathcal{T}_E$.

– Truyền thông điệp quan hệ. Tương tự, các biến thể quan hệ của các kiến trúc được trình bày khác cũng có thể được định nghĩa. Nhìn chung, chúng ta có thể mở rộng khuôn khổ truyền thông điệp

$$\mathbf{h}_i^{(l+1)} = \phi\left(\mathbf{h}_i^{(l)}, \bigoplus_{j\in\mathcal{N}_i}\psi^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)})\right)$$

để xử lý các đồ thị không đồng nhất chứa nhiều kiểu nút $\mathcal{T}_V$ & kiểu cạnh $r = (t_{\text{src}}, \text{rel}, t_{\text{dst}}) \in \mathcal{T}_E$.

Adaptation involves making core components type-aware. For a node i of type t_{dst}: [pp. 31–33]

- 4. Reinforcement Learning vs. Imitation Learning. Reinforcement learning & imitation learning represent 2 distinct paradigms for training agents to solve sequential decision-making problems e.g. job shop scheduling. Reinforcement learning aims to maximize a reward function (e.g., negative makespan). To achieve this objective, an agent (e.g., a GNN dispatcher) interacts with environment, learning from its mistakes. On other hand, imitation learning utilizes a dataset generated by an expert (e.g., a CP solver). Agent does not interact with environment but predicts what expert would have done in that particular situation. In both cases, ML models are trained employing small instances, hoping to generalize this learned knowledge to bigger ones.

  – Học tăng cường & học bắt chước đại diện cho 2 mô hình riêng biệt để huấn luyện tác tử giải quyết các vấn đề ra quyết định tuần tự, ví dụ như lập lịch xưởng. Học tăng cường hướng đến việc tối đa hóa hàm thưởng (ví dụ: khoảng thời gian hoàn thành âm). Để đạt được mục tiêu này, 1 tác tử (ví dụ: bộ điều phối GNN) tương tác với môi trường, học hỏi từ những sai lầm của nó. Mặt khác, học bắt chước sử dụng tập dữ liệu do chuyên gia tạo ra (ví dụ: bộ giải CP). Tác tử không tương tác với môi trường mà dự đoán những gì chuyên gia sẽ làm trong tình huống cụ thể đó. Trong cả 2 trường hợp, các mô hình ML được huấn luyện bằng cách sử dụng các trường hợp nhỏ, với hy vọng khái quát hóa kiến thức đã học được này thành các trường hợp lớn hơn.

  This chap also introduces Markov decision processes.. They are mathematical framework used to model sequential decision-making problems, including JSSP. This framework helps us explain components present in both reinforcement & imitation

learning. It is also foundation for Chap. 5, where we analyze how precious work has defined each element. This chap is essential to understand `JobShopLib`'s contribution, because our library aims to support experimenting with changes to these components.

– Chương này cũng giới thiệu về quy trình quyết định Markov. Chúng là khuôn khổ toán học được sử dụng để mô hình hóa các bài toán ra quyết định tuần tự, bao gồm cả JSSP. Khuôn khổ này giúp chúng tôi giải thích các thành phần có trong cả học tăng cường & học bắt chước. Nó cũng là nền tảng cho Chương 5, nơi chúng tôi phân tích cách công trình quý giá đã định nghĩa từng yếu tố. Chương này rất cần thiết để hiểu được đóng góp của `JobShopLib`, vì thư viện của chúng tôi hướng đến việc hỗ trợ việc thử nghiệm các thay đổi đối với các thành phần này.

This sect introduces foundational concepts of these approaches, with particular focus on methods typically employed to solve JSSP: policy gradient methods (RL) & behavioral cloning (IL). Understanding behavior cloning is especially relevant because it is learning method employed by experiments described in Chap. 8. Policy gradient methods are described to understand previous works & justify reward element of `JobShopLib`'s environment. While they are not necessary for understanding this project's experiments, they are an example of specific popular learning methods that can benefit from `JobShopLib`'s environment.

– Phần này giới thiệu các khái niệm cơ bản của các phương pháp này, đặc biệt tập trung vào các phương pháp thường được sử dụng để giải quyết JSSP: phương pháp gradient chính sách (RL) & nhân bản hành vi (IL). Việc hiểu về nhân bản hành vi đặc biệt quan trọng vì đây là phương pháp học được sử dụng bởi các thí nghiệm được mô tả trong Chương 8. Các phương pháp gradient chính sách được mô tả để hiểu các công trình trước đó & biện minh cho phần tử phần thưởng của môi trường `JobShopLib`. Mặc dù chúng không cần thiết để hiểu các thí nghiệm của dự án này, nhưng chúng là 1 ví dụ về các phương pháp học phổ biến cụ thể có thể được hưởng lợi từ môi trường `JobShopLib`.

- 4.1. Markov Decision Processes. To understand reinforcement learning, 1st need to introduce concept of a Markov Decision Process (MDP) (Bellman, 1957), which provides a mathematical framework for modeling sequential decision-making problems. An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where:
  - $\mathcal{S}$ is a set of states representing environment's configuration.
  - $\mathcal{A}$ is a set of actions agent can take.
  - $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, where $P(s'|s, a)$ represents probability of transitioning to state s' given that agent takes action a in state s.
  - $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is reward function, where $R(s, a, s')$ represents immediate reward received after transitioning from state s to state s' by taking action a.
  - $\gamma \in [0, 1]$ is discount factor that determines importance of future rewards.

  – Quy trình Quyết định Markov. Để hiểu về học tăng cường, trước tiên cần giới thiệu khái niệm Quy trình Quyết định Markov (MDP) (Bellman, 1957), cung cấp 1 khuôn khổ toán học để mô hình hóa các bài toán ra quyết định tuần tự. MDP được định nghĩa là 1 bộ $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ trong đó:
  - $\mathcal{S}$ là 1 tập hợp các trạng thái biểu diễn cấu hình của môi trường.
  - $\mathcal{A}$ là 1 tập hợp các hành động mà tác nhân có thể thực hiện.
  - $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ là hàm xác suất chuyển đổi, trong đó $P(s'|s, a)$ biểu diễn xác suất chuyển đổi sang trạng thái s' với điều kiện tác nhân thực hiện hành động a trong trạng thái s.
  - $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ là hàm phần thưởng, trong đó $R(s, a, s')$ biểu thị phần thưởng tức thời nhận được sau khi chuyển từ trạng thái s sang trạng thái s' bằng cách thực hiện hành động a.
  - $\gamma \in [0, 1]$ là hệ số chiết khấu xác định tầm quan trọng của phần thưởng trong tương lai.

  In context of job shop scheduling problem, state $s \in \mathcal{S}$ represents current partial schedule, including unscheduled operations & which machines are currently busy. Action $a \in \mathcal{A}$ corresponds to selecting next operation to schedule on an available machine. Transition function updates schedule deterministically based on chosen operation, & reward function provides feedback based on metrics e.g. makespan reduction or nay other custom objective.

  – Trong bối cảnh bài toán lập lịch xưởng, trạng thái $s \in \mathcal{S}$ biểu diễn lịch trình 1 phần hiện tại, bao gồm các hoạt động chưa được lên lịch & các máy hiện đang bận. Hành động $a \in \mathcal{A}$ tương ứng với việc chọn hoạt động tiếp theo để lên lịch trên 1 máy khả dụng. Hàm chuyển tiếp cập nhật lịch trình 1 cách xác định dựa trên hoạt động đã chọn, & hàm thưởng cung cấp phản hồi dựa trên các số liệu, ví dụ: giảm thời gian hoàn thành hoặc bất kỳ mục tiêu tùy chỉnh nào khác.

  Goal in an MDP: find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes expected cumulative discounted reward:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \right],$$

  where $\mathbb{E}_\pi$ denotes expected value when following policy π. Note $J(\pi)$ represents optimization objective that quantifies long-term performance of policy π from initial state distribution, accounting for both immediate rewards & future consequences of current decisions through discounting mechanism. Note also: this formula contains an implicit recurrence, as each state s_{t+1} is determined by current state s_t & action $\pi(s_t)$ according to transition probabilities P. This creates a chain of dependencies where each state influences all future states & rewards in sequence.

– Mục tiêu trong MDP: tìm 1 chính sách $\pi : \mathcal{S} \to \mathcal{A}$ tối đa hóa phần thưởng chiết khấu tích lũy kỳ vọng:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \right],$$

trong đó $\mathbb{E}_\pi$ biểu thị giá trị kỳ vọng khi tuân theo chính sách π. Lưu ý $J(\pi)$ biểu thị mục tiêu tối ưu hóa, định lượng hiệu suất dài hạn của chính sách π từ phân phối trạng thái ban đầu, tính đến cả phần thưởng tức thời & hậu quả tương lai của các quyết định hiện tại thông qua cơ chế chiết khấu. Lưu ý thêm: công thức này chứa 1 phép đệ quy ngầm định, vì mỗi trạng thái s_{t+1} được xác định bởi trạng thái hiện tại s_t & hành động $\pi(s_t)$ theo xác suất chuyển đổi P. Điều này tạo ra 1 chuỗi phụ thuộc trong đó mỗi trạng thái ảnh hưởng đến tất cả các trạng thái tương lai & phần thưởng theo trình tự.

A key characteristic of MDP is Markov property, which states: future state depends only on current state & action, not on history of previous states & actions.

– 1 đặc điểm chính của MDP là thuộc tính Markov, trong đó nêu: trạng thái tương lai chỉ phụ thuộc vào trạng thái hiện tại & hành động, không phụ thuộc vào lịch sử của các trạng thái & hành động trước đó.

∗ 4.1.1. Semi-Markov Decision Processes. While MDPs assume that actions take a uniform amount of time, JSSP is usually formulated in a way that involves actions with variable durations. A Semi-Markov Decision Process (SMDP) (Ross, 1992) extends MDPs to account for this temporal variability.

– Quy trình Quyết định Bán Markov. Trong khi MDP giả định rằng các hành động diễn ra trong 1 khoảng thời gian thống nhất, JSSP thường được xây dựng theo cách bao gồm các hành động có thời lượng thay đổi. Quy trình Quyết định Bán Markov (SMDP) (Ross, 1992) mở rộng MDP để tính đến sự biến thiên theo thời gian này.

An SMDP is defined similarly to an MDP but includes an additional component for duration of actions. Formally, an SMDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R, F, \gamma)$ where

· $\mathcal{S}, \mathcal{A}, P, R, \gamma$ are defined as in an MDP.

· $F : \mathcal{S} \times \mathcal{A} \times \mathcal{A} \times \mathbb{R}^+ \to [0, 1]$ is sojourn time distribution, which measures time spent in a state before transitioning to another state. I.e., $F(s, a, s', \tau)$ represents probability that state transition from s to s', under action a, takes time τ.

– SMDP được định nghĩa tương tự như MDP nhưng bao gồm 1 thành phần bổ sung cho thời lượng của các hành động. Về mặt hình thức, SMDP là 1 bộ $(\mathcal{S}, \mathcal{A}, P, R, F, \gamma)$ trong đó

· $\mathcal{S}, \mathcal{A}, P, R, \gamma$ được định nghĩa như trong MDP.

· $F : \mathcal{S} \times \mathcal{A} \times \mathcal{A} \times \mathbb{R}^+ \to [0, 1]$ là phân phối thời gian lưu trú, đo thời gian ở trong 1 trạng thái trước khi chuyển sang trạng thái khác. Tức là, $F(s, a, s', \tau)$ biểu diễn xác suất trạng thái chuyển đổi từ s sang s', dưới tác động a, mất thời gian τ.

Objective in an SMDP: find a policy that maximizes expected cumulative discounted reward. Only difference in practice with an MDP: discount factor needs to be adjusted to account for variable time between actions:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^{t_k} R(s_k, \pi(s_k), s_{k+1}) \right],$$

where t_k represents time at which kth decision is made.

○ 4.2. Reinforcement Learning. Reinforcement learning (Sutton & Barto, 2018) is a learning paradigm where an agent learns to make sequential decisions by interacting with an environment modeled as an MDP. Fundamental difference between RL & other ML approaches: RL involves learning from interaction rather than from labeled examples.

– Học tăng cường (Sutton & Barto, 2018) là 1 mô hình học tập trong đó 1 tác nhân học cách đưa ra quyết định tuần tự bằng cách tương tác với 1 môi trường được mô hình hóa như 1 MDP. Sự khác biệt cơ bản giữa RL & các phương pháp học máy khác: RL liên quan đến việc học từ tương tác hơn là từ các ví dụ được gắn nhãn.

In RL, agent does not have explicit knowledge of transition & reward functions but must learn them through experience. Learning process involves exploring environment, trying different actions, & observing resulting states & rewards. Goal: learn a policy that maximizes expected cumulative reward.

– Trong RL, tác nhân không có kiến thức rõ ràng về các hàm chuyển tiếp & phần thưởng mà phải học chúng thông qua kinh nghiệm. Quá trình học tập bao gồm việc khám phá môi trường, thử các hành động khác nhau, & quan sát các trạng thái & phần thưởng kết quả. Mục tiêu: học 1 chính sách tối đa hóa phần thưởng tích lũy kỳ vọng.

∗ 4.2.1. Value Functions. In reinforcement learning, agents learn through experience by interacting with their environment across many different states. While some RL approaches like policy gradient methods can directly optimize policies without explicit state evaluations, many powerful algorithms rely on concept of "value" – how good it is to be in a particular state or to take a specific action.

– Hàm Giá trị. Trong học tăng cường, các tác nhân học hỏi thông qua kinh nghiệm bằng cách tương tác với môi trường của chúng qua nhiều trạng thái khác nhau. Trong khi 1 số phương pháp RL như phương pháp gradient chính sách có thể tối ưu hóa trực tiếp các chính sách mà không cần đánh giá trạng thái rõ ràng, nhiều thuật toán mạnh mẽ lại dựa trên khái niệm "giá trị" – mức độ tốt khi ở trong 1 trạng thái cụ thể hoặc thực hiện 1 hành động cụ thể.

Value functions provide an intuitive way to evaluate decisions: a state is valuable if it leads to high immediate rewards plus access to other valuable future states. Mathematically, state-value function for policy π is denoted as

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) | s_0 = s \right].$$

Similarly, can define how good it is to take a specific action in a given state. Action-value function returns expected return after taking action a in state s & then following policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) | s_0 = s, a_0 = a \right].$$

– Hàm giá trị cung cấp 1 cách trực quan để đánh giá các quyết định: 1 trạng thái có giá trị nếu nó dẫn đến phần thưởng tức thời cao cộng với khả năng tiếp cận các trạng thái tương lai có giá trị khác. Về mặt toán học, hàm giá trị trạng thái cho chính sách π được ký hiệu là

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) | s_0 = s \right].$$

Tương tự, có thể xác định mức độ tốt khi thực hiện 1 hành động cụ thể trong 1 trạng thái nhất định. Hàm hành động-giá trị trả về lợi nhuận kỳ vọng sau khi thực hiện hành động a trong trạng thái s & sau đó tuân theo chính sách π:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) | s_0 = s, a_0 = a \right].$$

E.g., if know value associated with each state-action pair for our policy, a better policy can be derived by choosing action with highest associated value:

$$a(s) = \arg\max_a Q^\pi(s,a).$$

This is what many RL methods, e.g. Q-learning, do.
– Ví dụ, nếu biết giá trị liên kết với mỗi cặp trạng thái-hành động cho chính sách của chúng ta, 1 chính sách tốt hơn có thể được rút ra bằng cách chọn hành động có giá trị liên kết cao nhất:

$$a(s) = \arg\max_a Q^\pi(s,a).$$

Đây là cách mà nhiều phương pháp RL, ví dụ như Q-learning, thực hiện.
However, sometimes not necessary to calculate absolute value of expected return after taking action a, but rather to know relative advantage of taking that action. Advantage function calculates this relative improvement, which measures how much better it is to take action a compared to sampling an action following probability distribution given by policy $\pi(\cdot|s)$. Mathematically, this can be expressed as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

– Tuy nhiên, đôi khi không cần thiết phải tính giá trị tuyệt đối của lợi nhuận kỳ vọng sau khi thực hiện hành động a, mà cần biết lợi thế tương đối của việc thực hiện hành động đó. Hàm lợi thế tính toán mức cải thiện tương đối này, đo lường mức độ tốt hơn khi thực hiện hành động a so với việc lấy mẫu 1 hành động theo phân phối xác suất được cho bởi chính sách $\pi(\cdot|s)$. Về mặt toán học, điều này có thể được biểu thị như sau:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

∗ 4.2.2. Policy Gradient Methods. In practice, state & action spaces are often too large, or even continuous, making it infeasible to maintain a table of values for each state or state-action pair. Instead, function approximation methods are used to represent policy, value function, or action-value function. E.g., methods in Q-learning family attempt to approximate function $Q^*(s,a)$ associated with optimal policy using a parameterized function $Q_\theta(s,a)$, where θ are parameters of function (e.g., weights of a neural network).
– Phương pháp Gradient Chính sách. Trên thực tế, không gian trạng thái & hành động thường quá lớn, hoặc thậm chí liên tục, khiến việc duy trì bảng giá trị cho từng trạng thái hoặc cặp trạng thái-hành động trở nên bất khả thi. Thay vào đó, các phương pháp xấp xỉ hàm được sử dụng để biểu diễn chính sách, hàm giá trị hoặc hàm hành động-giá trị. Ví dụ: các phương pháp trong họ học Q cố gắng xấp xỉ hàm $Q^*(s,a)$ liên quan đến chính sách tối ưu bằng cách sử dụng hàm tham số $Q_\theta(s,a)$, trong đó θ là các tham số của hàm (ví dụ: trọng số của mạng nơ-ron).
In this sect, focus on methods that try to learn optimal policy directly because they are most used in practice for solving JSSP (Zhang et al., 2020; Park et al., 2021b,a; Ho et al., 2023). For this, policy is usually parameterized using a differentiable function $\pi_\theta(a|s)$ w.r.t. its parameters. Goal: find values of θ that maximize expected return $J(\pi_\theta)$. Specifically, would like to optimize policy using gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}.$$

– Trong phần này, chúng tôi tập trung vào các phương pháp cố gắng học chính sách tối ưu trực tiếp vì chúng được sử dụng nhiều nhất trong thực tế để giải JSSP (Zhang & cộng sự, 2020; Park & cộng sự, 2021b,a; Ho & cộng sự, 2023). Để làm được điều này, chính sách thường được tham số hóa bằng 1 hàm khả vi $\pi_\theta(a|s)$ theo các tham số của nó. Mục tiêu: tìm các giá trị của θ sao cho lợi nhuận kỳ vọng tối đa $J(\pi_\theta)$. Cụ thể, chúng tôi muốn tối ưu hóa chính sách bằng phương pháp tăng dần gradient:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}.$$

Policy gradient theorem (Sutton et al., 1999) provides us with a way to compute this gradient (i.e., how to adjust policy parameters to increase expected rewards). At its core, policy gradient theorem tells us we can improve a policy by making advantageous actions more likely & disadvantageous actions less likely, without needing to model how our policy changes affect environment's state distribution.

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \cdot Q^{\pi_\theta}(s,a)],$$

where d^π: state distribution induced by policy π & $Q^{\pi_\theta}(s,a)$ is expected return when taking action a in state s & following policy π_θ thereafter. This formula tells us to adjust policy parameters in proportion to how much an action change affects policy (gradient term) & how good action is (Q-value).

– Định lý gradient chính sách (Sutton & cộng sự, 1999) cung cấp cho chúng ta 1 cách để tính toán gradient này (tức là cách điều chỉnh các tham số chính sách để tăng phần thưởng kỳ vọng). Về bản chất, định lý gradient chính sách cho chúng ta biết rằng chúng ta có thể cải thiện chính sách bằng cách làm cho các hành động có lợi có khả năng xảy ra cao hơn & các hành động bất lợi có khả năng xảy ra thấp hơn, mà không cần phải mô hình hóa cách các thay đổi chính sách của chúng ta ảnh hưởng đến phân phối trạng thái của môi trường.

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \cdot Q^{\pi_\theta}(s,a)],$$

trong đó d^π: phân phối trạng thái được tạo ra bởi chính sách π & $Q^{\pi_\theta}(s,a)$ là lợi nhuận kỳ vọng khi thực hiện hành động a trong trạng thái s & theo chính sách π_θ sau đó. Công thức này cho chúng ta biết cách điều chỉnh các tham số chính sách theo tỷ lệ với mức độ ảnh hưởng của thay đổi hành động đến chính sách (thuật ngữ gradient) & mức độ tốt của hành động (giá trị Q).

Some of more commonly used policy gradient algorithms:

1. REINFORCE (Williams, 1992) is most basic policy gradient algorithm, which uses Monte Carlo estimates of return to update policy. After collecting a batch of N episodes, policy is updated with following formula:

   $$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \cdot G_t^i,$$

   where $G_t^i = \sum_{k=t}^{T_i} \gamma^{k-t} r_k^i$: observed return starting from step t in episode i, & T_i is length of episode i. While simple, REINFORCE suffers from high variance in gradient estimates.

   – REINFORCE (Williams, 1992) là thuật toán gradient chính sách cơ bản nhất, sử dụng ước tính lợi nhuận Monte Carlo để cập nhật chính sách. Sau khi thu thập 1 loạt N tập, chính sách được cập nhật theo công thức sau:

   $$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \cdot G_t^i,$$

   trong đó $G_t^i = \sum_{k=t}^{T_i} \gamma^{k-t} r_k^i$: lợi nhuận quan sát được bắt đầu từ bước t trong tập i, & T_i là độ dài của tập i. Mặc dù đơn giản, REINFORCE lại có phương sai cao trong ước tính gradient.
2. Advantage Actor-Critic (A2C) (Mnih et al., 2016) reduces variance by using a critic network to estimate advantage function $A(s,a)$, which measures how much better taking action a is compared to average action in state s:

   $$\nabla_\theta J(\pi_\theta) \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_\theta \log \pi_\theta(a_i|s_i) \cdot A(s_i,a_i),$$

   where B: batch size, & $A(s_i,a_i)$ is typically estimated as $r_i+\gamma V(s_i')-V(s_i)$ for a 1-step advantage, or using Generalized Advantage Estimation (Schulman et al., 2015) for multistep advantages.

   – Advantage Actor-Critic (A2C) (Mnih & cộng sự, 2016) làm giảm phương sai bằng cách sử dụng mạng lưới phê bình để ước tính hàm lợi thế $A(s,a)$, hàm này đo lường mức độ tốt hơn của hành động a so với hành động trung bình ở trạng thái s:

   $$\nabla_\theta J(\pi_\theta) \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_\theta \log \pi_\theta(a_i|s_i) \cdot A(s_i,a_i),$$

   trong đó B: kích thước lô, & $A(s_i,a_i)$ thường được ước tính là $r_i + \gamma V(s_i') - V(s_i)$ cho lợi thế 1 bước, hoặc sử dụng Ước tính Lợi thế Tổng quát (Schulman & cộng sự, 2015) cho lợi thế nhiều bước.
3. Proximal Policy Optimization (PPO) (Schulman et al., 2017) improves training stability by limiting policy update size through a clipped objective function:

   $$L^{\text{CLIP}}(\theta) = \frac{1}{B}\sum_{i=1}^{B} \min(r_i(\theta)A(s_i,a_i), \text{clip}(r_i(\theta), 1\pm\epsilon)A(s_i,a_i)) \text{ where } r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)}$$

   is probability ratio between new & old policies, B: batch size, & ϵ is a hyperparameter that limits policy change magnitude, typically 0.2.

– Tối ưu hóa Chính sách Gần (PPO) (Schulman & cộng sự, 2017) cải thiện tính ổn định của quá trình huấn luyện bằng cách giới hạn kích thước cập nhật chính sách thông qua hàm mục tiêu bị cắt bớt:

$$L^{\text{CLIP}}(\theta) = \frac{1}{B}\sum_{i=1}^{B} \min(r_i(\theta)A(s_i,a_i), \text{clip}(r_i(\theta), 1\pm\epsilon)A(s_i,a_i)) \text{ trong đó } r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)}$$

là tỷ lệ xác suất giữa chính sách mới & cũ, B: kích thước lô, & ϵ là 1 siêu tham số giới hạn mức độ thay đổi chính sách, thường là 0,2.

- 4.3. Imitation Learning. Unlike RL, which learns from environment feedback, IL IL (Hussein et al., 2017) learns a policy by mimicking demonstrations provided by an expert. In context of job shop scheduling, expert is typically a CP solver that produces optimal schedule (Lee & Kim, 2022).

  – Học bắt chước. Không giống như RL, học từ phản hồi môi trường, IL (Hussein & cộng sự, 2017) học 1 chính sách bằng cách bắt chước các minh họa do chuyên gia cung cấp. Trong bối cảnh lập lịch xưởng, chuyên gia thường là 1 bộ giải CP tạo ra lịch trình tối ưu (Lee & Kim, 2022).

  * 4.3.1. Behavioral Cloning. In this work, focus specifically on Behavioral Cloning (BC) (Pomerleau, 1988), which treats imitation learning as a supervised learning problem. Given a dataset of expert demonstrations [In our case, "expert" is a CP solver capable of obtaining optimal schedules for small problems.] $\mathcal{D}_E = \{(s_i, a_i^*)\}_{i=1}^N$ where a_i^* is action chosen by expert in state s_i, BC learns a policy π_θ by minimizing difference between policy's predictions & expert's actions:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(s,a^*)\sim\mathcal{D}_E}[L(\pi_\theta(s), a^*)],$$

  where L is a loss function measuring dissimilarity between predicted & expert actions. E.g., in experiments made in this project, GNN will learn to predict what actions are optimal for a given state s_k. I.e., it will try to learn a mapping $f : a_k \to [0,1]$ for $a_k \in \mathcal{A}(s_k)$, where $\mathcal{A}(s_k)$ is set of available actions at state s_k. Loss function employed in this case is binary cross-entropy:

$$L = -\sum_{O_{ij}\in\mathcal{A}(s_k)} y_{ij}\log\hat{y}_{ij} + (1-y_{ij})\log(1-\hat{y}_{ij})$$

  where y_{ij} is true label (0 or 1) for operation O_{ij}, $\hat{y}_{ij}$ is predicted probability that operation O_{ij} is optimal (belongs to class 1). This formula handles both possible outcomes: $L = -\log\hat{y}$ when $y = 1$, & $L = -\log(1-\hat{y})$ when $y = 0$.
  – Nhân bản hành vi. Trong công trình này, tập trung cụ thể vào Nhân bản hành vi (BC) (Pomerleau, 1988), coi việc học bắt chước là 1 vấn đề học có giám sát. Với 1 tập dữ liệu trình diễn của chuyên gia [Trong trường hợp của chúng tôi, "chuyên gia" là 1 bộ giải CP có khả năng thu được các lịch trình tối ưu cho các bài toán nhỏ.] $\mathcal{D}_E = \{(s_i, a_i^*)\}_{i=1}^N$ trong đó a_i^* là hành động được chuyên gia lựa chọn trong trạng thái s_i, BC học 1 chính sách π_θ bằng cách giảm thiểu sự khác biệt giữa dự đoán của chính sách & hành động của chuyên gia:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(s,a^*)\sim\mathcal{D}_E}[L(\pi_\theta(s), a^*)],$$

  trong đó L là hàm mất mát đo lường sự khác biệt giữa các hành động & hành động được dự đoán của chuyên gia. Ví dụ: trong các thí nghiệm được thực hiện trong dự án này, GNN sẽ học cách dự đoán hành động nào là tối ưu cho 1 trạng thái s_k nhất định. Tức là, nó sẽ cố gắng học 1 ánh xạ $f : a_k \to [0,1]$ cho $a_k \in \mathcal{A}(s_k)$, trong đó $\mathcal{A}(s_k)$ là tập hợp các hành động khả dụng tại trạng thái s_k. Hàm mất mát được sử dụng trong trường hợp này là entropy chéo nhị phân:

$$L = -\sum_{O_{ij}\in\mathcal{A}(s_k)} y_{ij}\log\hat{y}_{ij} + (1-y_{ij})\log(1-\hat{y}_{ij})$$

  trong đó y_{ij} là nhãn đúng (0 hoặc 1) cho phép toán O_{ij}, $\hat{y}_{ij}$ là xác suất dự đoán rằng phép toán O_{ij} là tối ưu (thuộc lớp 1). Công thức này xử lý cả 2 kết quả có thể xảy ra: $L = -\log\hat{y}$ khi $y = 1$, & $L = -\log(1-\hat{y})$ khi $y = 0$.
  1 of advantages of BC over RL: BC is more stable & simpler to train. While RL requires exploring different approaches, BC methods can directly learn from optimal actions. However, BC has limitations. BC can struggle with distributional shift – situations where learned policy encounters states not covered in training data. Additionally, expert demonstrations can be expensive to obtain for large problem instances where computing optimal solutions with exact solvers becomes intractable.
  – 1 trong những ưu điểm của BC so với RL: BC ổn định hơn & dễ huấn luyện hơn. Trong khi RL đòi hỏi phải khám phá các phương pháp tiếp cận khác nhau, các phương pháp BC có thể học trực tiếp từ các hành động tối ưu. Tuy nhiên, BC có những hạn chế. BC có thể gặp khó khăn với sự dịch chuyển phân phối - những tình huống mà chính sách đã học gặp phải các trạng thái không được đề cập trong dữ liệu huấn luyện. Ngoài ra, việc trình diễn chuyên gia có thể tốn kém đối với các trường hợp bài toán lớn, nơi việc tính toán các giải pháp tối ưu với các bộ giải chính xác trở nên khó khăn.

- 5. Literature Review & Problem Formulation. 1 of core objectives of this project: develop an open-source & flexible RL environment for solving JSSP sequentially. However, before describing its design, need to outline design space for modeling JSSP as a sequential decision-making problem. E.g., 1 of these choices is definition of operations available to be dispatched at each step. Some works consider only operations that can start immediately (Zhang et al., 2020; Park et al., 2021b), while others (Park et al., 2021a; Lee & Kim, 2022, 2024) allow the GNN dispatcher to reserve operations for later.

– 1 trong những mục tiêu cốt lõi của dự án này: phát triển 1 môi trường RL nguồn mở & linh hoạt để giải quyết JSSP tuần tự. Tuy nhiên, trước khi mô tả thiết kế của nó, cần phác thảo không gian thiết kế để mô hình hóa JSSP như 1 bài toán ra quyết định tuần tự. Ví dụ: 1 trong những lựa chọn này là định nghĩa các thao tác có thể được phân bổ tại mỗi bước. 1 số công trình chỉ xem xét các thao tác có thể bắt đầu ngay lập tức (Zhang & cộng sự, 2020; Park & cộng sự, 2021b), trong khi những công trình khác (Park & cộng sự, 2021a; Lee & Kim, 2022, 2024) cho phép bộ điều phối GNN dành riêng các thao tác cho sau này.

Once this design space is outlined, we will be in a position to abstract these choices into our RL environment. Following example above, all possible definitions of available actions at state s ($\mathcal{A}(s) \subseteq \mathcal{A}$) can be considered as application of a filter f to most flexible possible action space definition $\mathcal{A}'(s)$ ($\mathcal{A}(s) = f(\mathcal{A}'(s)) \subseteq \mathcal{A}'(s) \subseteq \mathcal{A}$).

– Khi không gian thiết kế này được phác thảo, chúng ta sẽ có thể trừu tượng hóa các lựa chọn này vào môi trường RL của mình. Theo ví dụ trên, tất cả các định nghĩa khả thi của các hành động khả dụng ở trạng thái s ($\mathcal{A}(s) \subseteq \mathcal{A}$) có thể được coi là ứng dụng của bộ lọc f cho định nghĩa không gian hành động linh hoạt nhất có thể $\mathcal{A}'(s)$ ($\mathcal{A}(s) = f(\mathcal{A}'(s)) \subseteq \mathcal{A}'(s) \subseteq \mathcal{A}$).

This broad design space stems from multiple ways we can define any of MDP components mentioned in prev chap. An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Some of these components are constant when solving classical JSSP described in Chap. 2. 1st, in case of classical JSSP, transition probability function P is deterministic & is common for every formulation. Scheduling an available operation on a machine is always successful, & its processing time is fixed. Similarly, probability distribution of time spent in a particular state before transitioning to another state F is also deterministic; transition time τ between kth state & next one is always same given a state s_k & action a_k.

– Không gian thiết kế rộng này bắt nguồn từ nhiều cách chúng ta có thể định nghĩa bất kỳ thành phần MDP nào được đề cập trong chương trước. 1 MDP là 1 bộ $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. 1 số thành phần này là hằng số khi giải JSSP cổ điển được mô tả trong Chương 2. Thứ nhất, trong trường hợp JSSP cổ điển, hàm xác suất chuyển đổi P là xác định & là chung cho mọi công thức. Lên lịch cho 1 hoạt động khả dụng trên máy luôn thành công, & thời gian xử lý của nó là cố định. Tương tự như vậy, phân phối xác suất của thời gian dành cho 1 trạng thái cụ thể trước khi chuyển sang trạng thái F khác cũng là xác định; thời gian chuyển đổi τ giữa trạng thái k & trạng thái tiếp theo luôn giống nhau với trạng thái s_k & hành động a_k.

1 customizable component is discount factor γ, which is typically set to 1 in studies that employ RL to solve standard JSSPs [It can be set to 1 safely because of lack of uncertainty in classical scenario & episodic nature of problem. No uncertainty means there is no need to prioritize short-term rewards over long-term ones. Additionally, since SMDP is episodic, there are no convergence issues by setting γ to 1.] (Zhang et al., 2020; Park et al., 2021b; Ho et al., 2023). Only exception is (Park et al., 2021a), which sets it to 0.9. However, this is usually considered an additional hyperparameter of RL agent rather than something hardcoded into environment. It also does not seem to have a considerable effect.

– 1 thành phần tùy chỉnh là hệ số chiết khấu γ, thường được đặt thành 1 trong các nghiên cứu sử dụng RL để giải quyết JSSP tiêu chuẩn [Có thể đặt thành 1 1 cách an toàn vì thiếu sự không chắc chắn trong kịch bản cổ điển & bản chất theo từng đợt của vấn đề. Không có sự không chắc chắn có nghĩa là không cần ưu tiên phần thưởng ngắn hạn hơn phần thưởng dài hạn. Ngoài ra, vì SMDP là theo từng đợt, nên không có vấn đề hội tụ nào khi đặt γ thành 1.] (Zhang & cộng sự, 2020; Park & cộng sự, 2021b; Ho & cộng sự, 2023). Ngoại lệ duy nhất là (Park & cộng sự, 2021a), đặt thành 0,9. Tuy nhiên, điều này thường được coi là 1 siêu tham số bổ sung của tác nhân RL hơn là thứ được mã hóa cứng vào môi trường. Nó cũng dường như không có tác dụng đáng kể.

However, previous works differ significantly on remaining components that define JSSP's SMDP. These components are state representation, definition of available actions for each state, & reward function. Depiction of many of these components can be critical. E.g., state transition proposed by Park et al. (2021b) impedes applications in real-time environments because dispatching an operation can modify start times of already scheduled ones.

– Tuy nhiên, các nghiên cứu trước đây có sự khác biệt đáng kể về các thành phần còn lại định nghĩa SMDP của JSSP. Các thành phần này bao gồm biểu diễn trạng thái, định nghĩa các hành động khả dụng cho mỗi trạng thái & hàm thưởng. Việc mô tả nhiều thành phần này có thể rất quan trọng. Ví dụ: chuyển đổi trạng thái do Park & cộng sự đề xuất (2021b) gây cản trở cho các ứng dụng trong môi trường thời gian thực vì việc phân phối 1 hoạt động có thể thay đổi thời gian bắt đầu của các hoạt động đã được lên lịch.

In this chap, analyze different choices of these elements made by prev papers. This analysis allows us to devise all features that a customizable RL environment for JSSP must have. Additionally, review other RL environments, showing how they fail to meet identified requirements.

– Trong chương này, chúng ta sẽ phân tích các lựa chọn khác nhau về các yếu tố này được thực hiện bởi các bài báo trước. Phân tích này cho phép chúng ta thiết kế tất cả các tính năng mà 1 môi trường RL tùy chỉnh cho JSSP cần có. Ngoài ra, chúng ta cũng sẽ xem xét các môi trường RL khác, chỉ ra cách chúng không đáp ứng được các yêu cầu đã xác định.

- 5.1. State Representation. Focus on graph representations of environment in this project. Therefore, a state is represented by a dynamic graph $G_k = (V_k, E_k, \mathbf{X}_k)$. This graph definition expands one presented in Sect. 2.2 by introducing a node feature matrix X_k & a step index k. Use k instead of t to differentiate it from current time of schedule, denoted by t_k.

  –Biểu diễn Trạng thái. Tập trung vào biểu diễn đồ thị của môi trường trong dự án này. Do đó, 1 trạng thái được biểu diễn bằng 1 đồ thị động $G_k = (V_k, E_k, \mathbf{X}_k)$. Định nghĩa đồ thị này mở rộng định nghĩa được trình bày trong Phần 2.2 bằng cách giới thiệu 1 ma trận đặc trưng nút X_k & 1 chỉ số bước k. Sử dụng k thay vì t để phân biệt với thời điểm hiện tại của lịch trình, được ký hiệu là t_k.

In this sect, 1st look at main definition variants of V_k, E_k. In Subsects. 5.1.1–5.1.2, explore 2 main graph variants: disjunctive & resource-task graphs, resp. In Subsect. 5.1.3, expand on different ways that these graphs can be updated after each step by introducing concept of residual scheduling (introduced in Lee & Kim (2022) & Ho et al. (2023)). Finally, in Subsection 5.1.4, mention some of features typically embedded into feature matrix X_k & principles that all features should follow, as stated by Ho et al. (2023) & Lee & Kim (2024).

– Trong phần này, trước tiên hãy xem xét các biến thể định nghĩa chính của V_k, E_k. Trong các Tiểu mục 5.1.1–5.1.2, hãy khám phá 2 biến thể đồ thị chính: đồ thị rời rạc & đồ thị tài nguyên-nhiệm vụ, tương ứng. Trong Tiểu mục 5.1.3, hãy mở rộng các cách khác nhau để cập nhật các đồ thị này sau mỗi bước bằng cách giới thiệu khái niệm lập lịch dư (được giới thiệu trong Lee & Kim (2022) & Ho & cộng sự (2023)). Cuối cùng, trong Tiểu mục 5.1.4, hãy đề cập đến 1 số đặc trưng thường được nhúng vào ma trận đặc trưng X_k & các nguyên tắc mà tất cả các đặc trưng nên tuân theo, như đã được Ho & cộng sự (2023) & Lee & Kim (2024) nêu ra.

∗ 5.1.1. Disjunctive Graphs. 1st approach for encoding problem into a graph uses disjunctive graph without dummy nodes (source & sink nodes) (Park et al., 2021b). After each step, dynamic information of schedule (e.g., if an operation has been scheduled) needs to be updated. This information is typically added by removing or adding disjunctive arcs & updating node features representing state of operation. Direction of disjunctive arc represents scheduling decisions. E.g., if operation O_{11} is scheduled before O_{22}, then there is an arc $O_{11} \rightarrow O_{22}$. See Fig. 2.3 for an example of a complete schedule represented by disjunctive graph. In Zhang et al. (2020), 2 graph updating strategies are defined:

· **"Adding-arc" strategy.** Graph starts with no disjunctive arcs. After each decision, corresponding disjunctive arc is added. This is approach used by Zhang et al. (2020). It has advantage of creating sparser graphs, which are more computationally efficient for GNN-based processing. However, an important limitation is loss of valuable information.

· **"Removing–arc" strategy.** Graph is initialized with disjunctive arcs pointing in both directions, & operation nodes that belong to same machine form a clique. After an operation is dispatched, 1 of 2 arcs that connected it with previous operation is dropped. Similarly, disjunctive arcs associated with previous scheduled operation are removed. This strategy, followed by Park et al. (2021b) & Lee & Kim (2024), has given better results than the previous one.

– 5.1.1. Đồ thị phân ly. Cách tiếp cận đầu tiên để mã hóa bài toán thành đồ thị sử dụng đồ thị phân ly không có nút giả (nút nguồn & nút đích) (Park & cộng sự, 2021b). Sau mỗi bước, thông tin động của lịch trình (ví dụ: nếu 1 thao tác đã được lên lịch) cần được cập nhật. Thông tin này thường được thêm vào bằng cách xóa hoặc thêm các cung phân ly & cập nhật các đặc trưng nút biểu diễn trạng thái thao tác. Hướng của cung phân ly biểu diễn các quyết định lập lịch trình. Ví dụ: nếu thao tác O_{11} được lên lịch trước O_{22}, thì sẽ có 1 cung $O_{11} \rightarrow O_{22}$. Xem Hình 2.3 để biết ví dụ về 1 lịch trình hoàn chỉnh được biểu diễn bằng đồ thị phân ly. Trong Zhang & cộng sự. (2020), 2 chiến lược cập nhật đồ thị được định nghĩa:

· **"Chiến lược thêm cung".** Đồ thị bắt đầu mà không có cung rời rạc. Sau mỗi quyết định, cung rời rạc tương ứng được thêm vào. Đây là phương pháp được Zhang & cộng sự (2020) sử dụng. Phương pháp này có ưu điểm là tạo ra các đồ thị thưa thớt hơn, hiệu quả tính toán hơn cho việc xử lý dựa trên GNN. Tuy nhiên, 1 hạn chế quan trọng là mất thông tin có giá trị.

· **"Chiến lược xóa cung".** Đồ thị được khởi tạo với các cung rời rạc hướng theo cả 2 hướng, & các nút thao tác thuộc cùng 1 máy tạo thành 1 nhóm. Sau khi 1 thao tác được phân phối, 1 trong 2 cung kết nối nó với thao tác trước đó sẽ bị loại bỏ. Tương tự, các cung rời rạc liên quan đến thao tác đã lên lịch trước đó sẽ bị loại bỏ. Chiến lược này, được Park & cộng sự (2021b) & Lee & Kim (2024) áp dụng, đã cho kết quả tốt hơn so với chiến lược trước đó.

Adding Extra Edges. Main limitation of standard disjunctive graphs used y Zhang et al. (2020): operations of same job are not well connected. Conjunctive edge always points to future operations (i.e., the one that has to be scheduled after current one). I.e., operations in which we are the most interested (1st operations of each job) do not receive messages directly from future operations of same job. For this reason, artificial conjunctive edges are typically added in other works. E.g., in Park et al. (2021b), an additional edge that connects each operation with its consecutive predecessor is considered. Additionally, they consider each of these edges to be of a different type & are processed independently:

$$h_i^{(l)} = \mathrm{MLP}_n^{(l)}\left(\left[\mathrm{ReLU}\left(\mathrm{MLP}_p^{(l)}\left(\sum_{j\in\mathcal{N}_v^p} h_j^{(l-1)}\right)\right);\mathrm{ReLU}\left(\mathrm{MLP}_s^{(l)}\left(\sum_{j\in\mathcal{N}_v^s} h_j^{(l-1)}\right)\right);\mathrm{ReLU}\left(\mathrm{MLP}_d^{(l)}\left(\sum_{j\in\mathcal{N}_v^d} h_j^{(l-1)}\right)\right);\mathrm{ReLU}\right.\right.$$

where $\mathrm{ReLU}(x) = \max\{0, x\} = x_+, \mathcal{N}_i^d, \mathcal{N}_i^p, \mathcal{N}_i^s$ represent set of disjunctive, precedent, & successor nodes, resp.; & $[;]$ is concatenation operation. They also connect each node with all nodes in graph, including itself, by aggregating them with a sum. Since this general vector is the same for all nodes, it only needs to be computed once.

– **Thêm Cạnh Bổ Sung.** Hạn chế chính của đồ thị phân ly chuẩn được sử dụng bởi Zhang & cộng sự (2020): các thao tác của cùng 1 công việc không được kết nối tốt. Cạnh liên hợp luôn trỏ đến các thao tác trong tương lai (tức là thao tác phải được lên lịch sau thao tác hiện tại). Tức là, các thao tác mà chúng ta quan tâm nhất (thao tác đầu tiên của mỗi công việc) không nhận được thông điệp trực tiếp từ các thao tác trong tương lai của cùng 1 công việc. Vì lý do này, các cạnh liên hợp nhân tạo thường được thêm vào trong các công việc khác. Ví dụ, trong Park & cộng sự (2021b), 1 cạnh bổ sung kết nối mỗi thao tác với thao tác tiền nhiệm liên tiếp của nó đã được xem xét. Ngoài ra, họ coi mỗi cạnh này là 1 loại khác nhau & được xử lý độc lập:

$$h_i^{(l)} = \mathrm{MLP}_n^{(l)}\left(\left[\mathrm{ReLU}\left(\mathrm{MLP}_p^{(l)}\left(\sum_{j\in\mathcal{N}_v^p} h_j^{(l-1)}\right)\right);\mathrm{ReLU}\left(\mathrm{MLP}_s^{(l)}\left(\sum_{j\in\mathcal{N}_v^s} h_j^{(l-1)}\right)\right);\mathrm{ReLU}\left(\mathrm{MLP}_d^{(l)}\left(\sum_{j\in\mathcal{N}_v^d} h_j^{(l-1)}\right)\right);\mathrm{ReLU}\right.\right.$$

trong đó $\text{ReLU}(x) = \max\{0, x\} = x_+$, $\mathcal{N}_i^d$, $\mathcal{N}_i^p$, $\mathcal{N}_i^s$ biểu diễn tập hợp các nút rời rạc, nút tiền lệ, & nút kế tiếp, tương ứng; & $[;]$ là phép toán nối. Chúng cũng kết nối mỗi nút với tất cả các nút trong đồ thị, bao gồm cả chính nó, bằng cách tổng hợp chúng bằng 1 phép toán tổng. Vì vectơ tổng quát này giống nhau cho tất cả các nút, nên nó chỉ cần được tính toán 1 lần.

Study that best exploits adding extra edges is Lee & Kim (2024). In addition to considering aforementioned neighborhoods, they connect each operation with all its successors & predecessors to "effectively deliver information between nodes."

– Nghiên cứu khai thác tốt nhất việc thêm các cạnh bổ sung là của Lee & Kim (2024). Ngoài việc xem xét các vùng lân cận đã đề cập, họ kết nối mỗi hoạt động với tất cả các hoạt động kế thừa & tiền nhiệm của nó để "truyền tải thông tin hiệu quả giữa các nút."

∗ 5.1.2. Resource-Task Graphs. Park et al. (2021a) introduced 2nd type of representation, which Ho et al. (2023) also used. They added nodes representing machines & removed disjunctive edges. While literature only presents this basic version with machine nodes, it can also be easily extended to include job & global nodes. In these extended variants, conjunctive edges can also be removed. Here, present 3 possible variants of resource-task graphs:

1. **Basic Resource-Task Graph.** Contains operation nodes & machine nodes. Machines are connected to their respective operations, & machine nodes form a clique (fully connected graph). Operations of same job are also connected through a structure similar to a machine clique. This is representation presented in Park et al. (2021a).
2. **Resource-Task Graph with Jobs.** Extends basic version by adding job nodes. In this representation, job nodes are connected to their respective operations, & jobs nodes also form a clique among themselves. This variant is a novel contribution not found in literature.
3. **Complete Resource-Task Graph.** Further extends representation by adding a global node that connects to all job & machine nodes, serving as an information aggregator. This variant is also a novel contribution.

– Đồ thị Tài nguyên-Nhiệm vụ. Park & cộng sự (2021a) đã giới thiệu loại biểu diễn thứ 2, mà Ho & cộng sự (2023) cũng đã sử dụng. Họ đã thêm các nút biểu diễn máy móc & loại bỏ các cạnh rời rạc. Mặc dù tài liệu chỉ trình bày phiên bản cơ bản này với các nút máy móc, nhưng nó cũng có thể dễ dàng được mở rộng để bao gồm các nút công việc & toàn cục. Trong các biến thể mở rộng này, các cạnh nối cũng có thể được loại bỏ. Dưới đây, chúng tôi trình bày 3 biến thể khả thi của đồ thị tài nguyên-nhiệm vụ:

1. **Đồ thị Tài nguyên-Nhiệm vụ Cơ bản.** Chứa các nút thao tác & nút máy. Các máy móc được kết nối với các thao tác tương ứng của chúng, & nút máy tạo thành 1 clique (đồ thị kết nối đầy đủ). Các thao tác của cùng 1 công việc cũng được kết nối thông qua 1 cấu trúc tương tự như 1 clique máy móc. Đây là biểu diễn được trình bày trong Park & cộng sự (2021a). ... Trong biểu diễn này, các nút công việc được kết nối với các thao tác tương ứng của chúng, & các nút công việc cũng tạo thành 1 nhóm nhỏ với nhau. Biến thể này là 1 đóng góp mới chưa được tìm thấy trong tài liệu.
2. **Đồ thị Tài nguyên-Nhiệm vụ Hoàn chỉnh.** Biểu diễn này được mở rộng hơn nữa bằng cách thêm 1 nút toàn cục kết nối với tất cả các nút công việc & máy, đóng vai trò là 1 bộ tổng hợp thông tin. Biến thể này cũng là 1 đóng góp mới.

For a problem with $|\mathcal{J}|$ jobs & $|\mathcal{M}|$ machines (resulting in $|\mathcal{J}| \times |\mathcal{M}|$ operations), resource-task graph approach offers computational advantages over disjunctive graphs for GNN processing. On 1 hand, assuming no recirculation of operations, disjunctive graphs have $O(|\mathcal{J}|^2 \times |\mathcal{M}|)$ disjunctive edges in worst case since each machine can have up to $|\mathcal{J}|$ operations, forming a clique with $O(|\mathcal{J}|^2)$ edges, across $|\mathcal{M}|$ machines.

– Đối với bài toán có $|\mathcal{J}|$ công việc & $|\mathcal{M}|$ máy (tạo ra $|\mathcal{J}| \times |\mathcal{M}|$ phép toán), phương pháp đồ thị tài nguyên-nhiệm vụ mang lại lợi thế tính toán so với đồ thị rời rạc trong xử lý GNN. Mặt khác, giả sử không có sự tuần hoàn của các phép toán, đồ thị rời rạc có $O(|\mathcal{J}|^2 \times |\mathcal{M}|)$ cạnh rời rạc trong trường hợp xấu nhất vì mỗi máy có thể có tối đa $|\mathcal{J}|$ phép toán, tạo thành 1 nhóm với $O(|\mathcal{J}|^2)$ cạnh, trên $|\mathcal{M}|$ máy.

Resource-task graphs reduce this number to $O(|\mathcal{J}| \times |\mathcal{M}|)$ edges between operations & resources, plus additional edges depending on variant chosen. These additional edges can be $O(|\mathcal{J}|^2)$ & $O(|\mathcal{M}|^2)$ if there are machine or job node cliques, resp. Even in these cases, however, complexity is reduced for graphs with $|\mathcal{J}| \leq |\mathcal{M}|^2$. For large job shop problems, reducing edge count can significantly improve computational efficiency while maintaining or enhancing graph's expressive power by explicitly modeling resource relationships.

– Đồ thị tài nguyên-nhiệm vụ giảm số này xuống còn $O(|\mathcal{J}| \times |\mathcal{M}|)$ cạnh giữa các thao tác & tài nguyên, cộng với các cạnh bổ sung tùy thuộc vào biến thể được chọn. Các cạnh bổ sung này có thể là $O(|\mathcal{J}|^2)$ & $O(|\mathcal{M}|^2)$ nếu có các nhóm nút máy hoặc công việc, tương ứng. Tuy nhiên, ngay cả trong những trường hợp này, độ phức tạp cũng giảm đối với đồ thị có $|\mathcal{J}| \leq |\mathcal{M}|^2$. Đối với các bài toán cửa hàng công việc lớn, việc giảm số lượng cạnh có thể cải thiện đáng kể hiệu quả tính toán trong khi vẫn duy trì hoặc tăng cường sức mạnh biểu đạt của đồ thị bằng cách mô hình hóa rõ ràng các mối quan hệ tài nguyên.

However, there are many differences between Park et al. (2021a) & Ho et al. (2023) with other approaches. These differences include SMDP formulation, model architecture used, learning method employed for training, & which extra edges were added to disjunctive graph. Therefore, graph representation's impact on performance is unclear.

– Tuy nhiên, có nhiều điểm khác biệt giữa Park & cộng sự (2021a) & Ho & cộng sự (2023) với các phương pháp tiếp cận khác. Những điểm khác biệt này bao gồm công thức SMDP, kiến trúc mô hình được sử dụng, phương pháp học được sử dụng để huấn luyện, & những cạnh bổ sung nào được thêm vào đồ thị rời rạc. Do đó, tác động của biểu diễn đồ thị lên hiệu suất vẫn chưa rõ ràng.

Additionally, job & global node variants were introduced theoretically in this work as possible future directions, but their effectiveness has yet to be proved with a rigorous ablation study. E.g., Complete Resource-Task Graph might suffer from

"over-squashing" (Alon & Yahav, 2021) because all information between machines or jobs must flow through single global node, creating a potential bottleneck in message passing.

– Ngoài ra, các biến thể công việc & nút toàn cục đã được giới thiệu về mặt lý thuyết trong nghiên cứu này như những hướng đi khả thi trong tương lai, nhưng hiệu quả của chúng vẫn chưa được chứng minh bằng 1 nghiên cứu cắt bỏ nghiêm ngặt. Ví dụ: Đồ thị Tài nguyên-Nhiệm vụ Hoàn chỉnh có thể bị "quá tải" (Alon & Yahav, 2021) vì tất cả thông tin giữa các máy hoặc công việc phải truyền qua 1 nút toàn cục duy nhất, tạo ra nguy cơ tắc nghẽn trong việc truyền tin nhắn.

Introduce these variants to note how a flexible RL environment should support heterogeneous graph representations with different types of nodes & edges.

– Giới thiệu các biến thể này để lưu ý cách môi trường RL linh hoạt hỗ trợ các biểu diễn đồ thị không đồng nhất với các loại nút & cạnh khác nhau.

* 5.1.3. Residual Scheduling. An additional mechanism for updating graph (Lee & Kim, 2022; Ho et al., 2023) has become standard. It consists of removing completed nodes & adjusting their features accordingly. Key idea comes from Markov property mentioned in Chap. 4: future state depends only on current state & action, not on history of previous ones. E.g., duration of completed operation that preceded current available one does not matter when deciding whether to dispatch it. Even scalar value of current time is irrelevant. Best schedule will be the same regardless of the hour of the day, for instance. Need to know this information for reconstructing schedule once have scheduled each operation, but not for representing state.

– Lập lịch Dư. 1 cơ chế bổ sung để cập nhật đồ thị (Lee & Kim, 2022; Ho & cộng sự, 2023) đã trở thành tiêu chuẩn. Cơ chế này bao gồm việc loại bỏ các nút đã hoàn thành & điều chỉnh các đặc trưng của chúng cho phù hợp. Ý tưởng chính xuất phát từ tính chất Markov được đề cập trong Chương 4: trạng thái tương lai chỉ phụ thuộc vào trạng thái hiện tại & hành động, chứ không phụ thuộc vào lịch sử của các hành động trước đó. Ví dụ, thời lượng của hoạt động đã hoàn thành trước hoạt động hiện tại không quan trọng khi quyết định có nên phân phối hay không. Ngay cả giá trị vô hướng của thời gian hiện tại cũng không liên quan. Ví dụ, lịch trình tốt nhất sẽ giống nhau bất kể giờ nào trong ngày. Cần biết thông tin này để xây dựng lại lịch trình sau khi đã lên lịch cho từng hoạt động, nhưng không phải để biểu diễn trạng thái.

Thus, removing completed nodes does not remove relevant information, helps prevent overfitting, & reduces computation during network's forward & backward passes. Similarly, adjusting node features creates a more accurate representation of current state. 1 example would be updating scheduled operations' duration to consider only their remaining processing time. E.g., in state presented in Fig. 5.2: Example of a disjunctive graph (b) representing a partial schedule (a) of instance defined in Table 1. Operation O_{33} is scheduled before O_{21} &, thus, has direction of their disjunctive edge defined. Nodes representing completed nodes have been removed. Red dotted line in (a) represents current time, defined as earliest start time of available operations. These operations are shown in legend, where "m" & "j" represent their machine or job id, resp. (starting from 0), "d" represents their duration or processing time p_{ij}, & "p" represents their position or index in job (starting from 0). processing time of O_{32} should be adjusted from 3 to 2. I.e., that situation would be equivalent to just starting an operation with a duration of 2 at current time. This strategy was also employed in Lee & Kim (2024).

– Do đó, việc loại bỏ các nút đã hoàn thành không loại bỏ thông tin có liên quan, giúp ngăn ngừa quá khớp, & giảm tính toán trong quá trình chuyển tiếp & lùi của mạng. Tương tự, việc điều chỉnh các đặc điểm của nút tạo ra biểu diễn chính xác hơn về trạng thái hiện tại. 1 ví dụ sẽ là cập nhật thời lượng của các hoạt động đã lên lịch để chỉ xem xét thời gian xử lý còn lại của chúng. Ví dụ: trong trạng thái được trình bày trong Hình 5.2: Ví dụ về đồ thị rời rạc (b) biểu diễn lịch trình 1 phần (a) của phiên bản được định nghĩa trong Bảng 1. Hoạt động O_{33} được lên lịch trước O_{21} &, do đó, có hướng của cạnh rời rạc của chúng được xác định. Các nút biểu diễn các nút đã hoàn thành đã bị loại bỏ. Đường chấm màu đỏ trong (a) biểu diễn thời gian hiện tại, được định nghĩa là thời gian bắt đầu sớm nhất của các hoạt động khả dụng. Các hoạt động này được hiển thị trong chú giải, trong đó "m" & "j" biểu diễn id máy hoặc công việc của chúng, tương ứng. (bắt đầu từ 0), "d" biểu thị thời lượng hoặc thời gian xử lý p_{ij} của chúng, & "p" biểu thị vị trí hoặc chỉ số của chúng trong công việc (bắt đầu từ 0). thời gian xử lý của O_{32} nên được điều chỉnh từ 3 thành 2. Tức là, tình huống đó sẽ tương đương với việc chỉ bắt đầu 1 hoạt động có thời lượng là 2 tại thời điểm hiện tại. Chiến lược này cũng được sử dụng trong Lee & Kim (2024).

* 5.1.4. Feature Matrix. Feature matrix X_k contains relevant information representing each node. E.g., 1 of attributes used by all aforementioned studies is processing time of each operation p_{ij}, or remaining processing time $p_{ij}^{(k)} = C_{ij} - t_k$ at s_k if using residual scheduling. Work that introduces most informative features at time of writing is (Lee & Kim, 2024). Some of these features are:
  · Ready time to start of operation O_{ij} at step k, denoted as $R_{ij}^{(k)}$. This feature is earliest start time $S_{ij}^{*(k)}$ possible that respects problem constraints adjusted by current time: $R_{ij}^{(k)} = S_{ij}^{*(k)} - t_k$. This adjustment is made because ongoing operations can be considered just started.
  · Machine load $L_{ij}^{(k)}$ of machine M_u associated with operation O_{ij}. This load is computed as sum of all remaining processing times of operations associated with machine M_u. It can then be normalized by total sum of remaining processing times of problem: $\sum_{O_{i'j'} \in \mathcal{O}^{(k)}} p_{i'j'}^{(k)}$ where $\mathcal{O}^{(k)}$ is set of uncompleted operations at step k.
  · Number of remaining operations in job J_i of operation O_{ij}.

– Ma trận Đặc trưng. Ma trận đặc trưng X_k chứa thông tin liên quan biểu diễn từng nút. Ví dụ: 1 trong các thuộc tính được sử dụng bởi tất cả các nghiên cứu đã đề cập ở trên là thời gian xử lý của mỗi thao tác p_{ij}, hoặc thời gian xử lý còn lại $p_{ij}^{(k)} = C_{ij} - t_k$ tại s_k nếu sử dụng lập lịch dư thừa. Công trình giới thiệu nhiều đặc điểm hữu ích nhất tại thời điểm viết là (Lee & Kim, 2024). 1 số đặc điểm này là:

  · Thời gian sẵn sàng để bắt đầu thao tác O_{ij} tại bước k, được ký hiệu là $R_{ij}^{(k)}$. Đặc điểm này là thời điểm bắt đầu sớm

nhất $S_{ij}^{*(k)}$ có thể tuân thủ các ràng buộc của bài toán được điều chỉnh theo thời gian hiện tại: $R_{ij}^{(k)} = S_{ij}^{*(k)} - t_k$. Việc điều chỉnh này được thực hiện vì các thao tác đang diễn ra có thể được coi là vừa mới bắt đầu.

· Tải máy $L_{ij}^{(k)}$ của máy M_u liên quan đến thao tác O_{ij}. Tải này được tính bằng tổng thời gian xử lý còn lại của các thao tác liên quan đến máy M_u. Sau đó, tải này có thể được chuẩn hóa bằng tổng thời gian xử lý còn lại của bài toán: $\sum_{O_{i'j'} \in \mathcal{O}^{(k)}} p_{i'j'}^{(k)}$ trong đó $\mathcal{O}^{(k)}$ là tập hợp các thao tác chưa hoàn thành tại bước k.

· Số thao tác còn lại trong công việc J_i của thao tác O_{ij}.

All features they extracted respected reasoning behind residual scheduling. Additionally, well-defined features should also respect symmetry of problem; they must not depend on arbitrary indices used to label machines or jobs. Another important consideration: scheduling decisions should remain constant between states to apply learned dispatcher in real-time settings. E.g., in Zhang et al. (2020) they adjust previous scheduling decisions to keep tight schedules. While effective in reducing makespan, this strategy impedes constructing schedule sequentially because start times are not fixed.

– Tất cả các đặc điểm được trích xuất đều tôn trọng lý luận đằng sau việc lập lịch dư thừa. Ngoài ra, các đặc điểm được xác định rõ ràng cũng phải tôn trọng tính đối xứng của vấn đề; chúng không được phụ thuộc vào các chỉ số tùy ý được sử dụng để dán nhãn máy móc hoặc công việc. 1 cân nhắc quan trọng khác: các quyết định lập lịch phải được duy trì không đổi giữa các trạng thái để áp dụng bộ điều phối đã học trong môi trường thời gian thực. Ví dụ, trong nghiên cứu của Zhang & cộng sự (2020), họ điều chỉnh các quyết định lập lịch trước đó để duy trì lịch trình chặt chẽ. Mặc dù hiệu quả trong việc giảm thời gian chờ, chiến lược này lại cản trở việc xây dựng lịch trình tuần tự vì thời gian bắt đầu không cố định.

◦ 5.2. Action Definition. Initial methodologies for utilizing dispatching rules in scheduling, e.g. those presented by Zhang et al. (2020) & Park et al. (2021b), adopted a strategy where only operations that can start at current time can be selected. They copy definition employed by dispatching rules defining scheduling decisions "when a machine becomes available". Consequently, this approach is limited to generating solely non-delay schedules as described in Sect. 2.1. In example shown in Table 5.1: Example of a JSSP problem with 2 jobs. Operations must be completed in order presented in table., impossible to obtain a solution different from the one shown on left of Fig. 5.3: Comparison of 2 possible solutions to instance defined in Table 5.1. Both solutions have been obtained by applying SPT dispatching rule. Difference: schedule (b) has been obtained by defining available operations as those that can start immediately. In contrast, schedule (a) has been created with a more flexible definition that allows reserving operations. The latter reaches optimal solution., regardless of dispatching rule used. This situations occurs because, at each step, there is only 1 available operation. Impossible to wait for operation O_{21} to be ready; instead, O_{22} must be dispatched immediately, resulting in a less efficient solution.

– Action Definition. Các phương pháp luận ban đầu để sử dụng các quy tắc điều phối trong lập lịch, ví dụ như các phương pháp do Zhang & cộng sự (2020) & Park & cộng sự (2021b) trình bày, đã áp dụng 1 chiến lược trong đó chỉ có thể chọn các hoạt động có thể bắt đầu tại thời điểm hiện tại. Họ sao chép định nghĩa được sử dụng bởi các quy tắc điều phối xác định các quyết định lập lịch "khi 1 máy trở nên khả dụng". Do đó, cách tiếp cận này bị giới hạn trong việc chỉ tạo các lịch trình không trì hoãn như được mô tả trong Phần 2.1. Trong ví dụ được hiển thị trong Bảng 5.1: Ví dụ về bài toán JSSP có 2 công việc. Các hoạt động phải được hoàn thành theo thứ tự được trình bày trong bảng., không thể có được giải pháp khác với giải pháp được hiển thị ở bên trái của Hình 5.3: So sánh 2 giải pháp khả thi cho phiên bản được xác định trong Bảng 5.1. Cả 2 giải pháp đều thu được bằng cách áp dụng quy tắc điều phối SPT. Sự khác biệt: lịch trình (b) thu được bằng cách xác định các hoạt động khả dụng là những hoạt động có thể bắt đầu ngay lập tức. Ngược lại, lịch trình (a) đã được tạo với định nghĩa linh hoạt hơn cho phép đặt trước các hoạt động. Giải pháp sau đạt được giải pháp tối ưu, bất kể quy tắc phân bổ được sử dụng. Tình huống này xảy ra vì tại mỗi bước, chỉ có 1 thao tác khả dụng. Không thể chờ thao tác O_{21} sẵn sàng; thay vào đó, O_{22} phải được phân bổ ngay lập tức, dẫn đến giải pháp kém hiệu quả hơn.

In contrast, other works (Park et al., 2021a; Lee & Kim, 2022, 2024) allow reserving operations. More flexible definition possible is to define an operation as ready to be dispatched if previous operation if its job has been scheduled. However, this definition can be improved by removing dominated operations from available action set (Lee & Kim, 2022). Consider an operation O_{ij} to be dominated by $O_{i'j'}$ if the latter can be completed before earliest start time of the 1st (i.e., if $C_{i'j'} \leq S_{ij}$) & both operations are ready to be processed on same machine.

– Ngược lại, các công trình khác (Park & cộng sự, 2021a; Lee & Kim, 2022, 2024) cho phép đặt trước các thao tác. 1 định nghĩa linh hoạt hơn có thể được định nghĩa là 1 thao tác sẵn sàng để được phân phối nếu thao tác trước đó đã được lên lịch. Tuy nhiên, định nghĩa này có thể được cải thiện bằng cách loại bỏ các thao tác bị chi phối khỏi tập hành động khả dụng (Lee & Kim, 2022). Hãy xem xét 1 thao tác O_{ij} bị chi phối bởi $O_{i'j'}$ nếu thao tác sau có thể được hoàn thành trước thời điểm bắt đầu sớm nhất của thao tác đầu tiên (tức là nếu $C_{i'j'} \leq S_{ij}$) & cả 2 thao tác đều sẵn sàng để được xử lý trên cùng 1 máy.

Excluding dominated operations acts as a heuristic that limits space of available actions to operations that can begin immediately. In fact, available actions definition used by Zhang et al. (2020) & Park et al. (2021b) can also be seen as a heuristic that filters operations that cannot start immediately.

– Việc loại trừ các hoạt động bị chi phối hoạt động như 1 phương pháp tìm kiếm giới hạn không gian hành động khả dụng thành các hoạt động có thể bắt đầu ngay lập tức. Trên thực tế, định nghĩa về hành động khả dụng được sử dụng bởi Zhang & cộng sự (2020) & Park & cộng sự (2021b) cũng có thể được xem như 1 phương pháp tìm kiếm lọc các hoạt động không thể bắt đầu ngay lập tức.

◦ 5.3. Reward Functions. Since objective is to minimize makespan, some studies have directly used negative makespan as reward $-C_{\max}$ (Park et al., 2021a; Infantes et al., 2024). This reward is defined as 0 for every step, except last one T, which has

negative makespan as reward:

$$R(s_{T_1}, a_{T-1}, s_T) = -t_T.$$

Note need to change makespan's sign to negative because reward is always maximized.

– Hàm Thưởng. Vì mục tiêu là giảm thiểu makespan, 1 số nghiên cứu đã trực tiếp sử dụng makespan âm làm phần thưởng $-C_{\max}$ (Park & cộng sự, 2021a; Infantes & cộng sự, 2024). Phần thưởng này được định nghĩa là 0 cho mọi bước, ngoại trừ bước cuối cùng T, có makespan âm làm phần thưởng:

$$R(s_{T_1}, a_{T-1}, s_T) = -t_T.$$

Lưu ý cần đổi dấu makespan thành âm vì phần thưởng luôn được tối đa hóa.

While objective focused on this work is makespan, simply returning makespan at end of an episode would result in a sparse reward. A sparse reward is one that occurs infrequently during an agent's learning process. This is problematic because agents must explore extensively before receiving any meaningful feedback, making it difficult to correlate actions with eventual rewards (credit assignment problem). This delays learning, decreases sample efficiency, & often leads to suboptimal policies as agent struggles to identify which actions contributed to success. As noted by Sutton & Barto (2018), dense reward shaping that provides more frequent feedback can help guide exploration & accelerate learning.

– Mặc dù mục tiêu tập trung vào công việc này là makespan, việc chỉ trả về makespan vào cuối mỗi tập sẽ dẫn đến phần thưởng thưa thớt. Phần thưởng thưa thớt là phần thưởng xuất hiện không thường xuyên trong quá trình học của tác nhân. Điều này gây ra vấn đề vì các tác nhân phải khám phá sâu rộng trước khi nhận được bất kỳ phản hồi có ý nghĩa nào, khiến việc liên hệ các hành động với phần thưởng cuối cùng trở nên khó khăn (vấn đề gán tín chỉ). Điều này làm chậm quá trình học, giảm hiệu quả lấy mẫu, & thường dẫn đến các chính sách không tối ưu khi tác nhân gặp khó khăn trong việc xác định hành động nào đã góp phần vào thành công. Như Sutton & Barto (2018) đã lưu ý, việc định hình phần thưởng dày đặc cung cấp phản hồi thường xuyên hơn có thể giúp định hướng quá trình khám phá & đẩy nhanh quá trình học.

Designs of dense reward functions have been very diverse. Zhang et al. (2020) define it as difference in quality measure between s_k, s_{k+1}:

$$R(s_k, a_k, s_{k+1}) = H(s_k) - H(s_{k+1}).$$

Here $H(s)$ represents quality measure of a state s, defined as lower bound of makespan $C_{\max}$ achievable from that state. This lower bound is computed as maximum estimated completion time considering only precedence constraints for unscheduled operations. With a discount factor of $\gamma = 1$, total cumulative rewards equals $H(s_0) - C_{\max}$. Since initial lower bound $H(s_0)$ is constant for a given problem instance, maximizing cumulative reward corresponds to minimizing final makespan.

– Các thiết kế hàm phần thưởng dày đặc rất đa dạng. Zhang & cộng sự (2020) định nghĩa nó là sự khác biệt về thước đo chất lượng giữa s_k, s_{k+1}:

$$R(s_k, a_k, s_{k+1}) = H(s_k) - H(s_{k+1}).$$

Ở đây $H(s)$ biểu diễn thước đo chất lượng của trạng thái s, được định nghĩa là giới hạn dưới của khoảng thời gian hoàn thành $C_{\max}$ có thể đạt được từ trạng thái đó. Giới hạn dưới này được tính là thời gian hoàn thành ước tính tối đa, chỉ xét đến các ràng buộc về thứ tự ưu tiên cho các hoạt động không theo lịch trình. Với hệ số chiết khấu $\gamma = 1$, tổng phần thưởng tích lũy bằng $H(s_0) - C_{\max}$. Vì giới hạn dưới ban đầu $H(s_0)$ là hằng số đối với 1 trường hợp bài toán nhất định, việc tối đa hóa phần thưởng tích lũy tương ứng với việc tối thiểu hóa khoảng thời gian hoàn thành cuối cùng.

Ho et al. (2023) simplify this reward by defining it as "a negative of additional processing time for dispatched action". However, not clear whether it represents

$$R(s_k, a_k, s_{k+1}) = t_k - t_{k+1}$$

or

$$R(s_k, a_k, a_{k+1}) = C_{\max}^{(k)} - C_{\max}^{(k+1)}.$$

In both interpretations, maximizing cumulative rewards is equivalent to minimizing makespan.

– Ho & cộng sự (2023) đã đơn giản hóa phần thưởng này bằng cách định nghĩa nó là " 1 giá trị âm của thời gian xử lý bổ sung cho hành động được phân bổ". Tuy nhiên, không rõ liệu nó biểu thị

$$R(s_k, a_k, s_{k+1}) = t_k - t_{k+1}$$

hay

$$R(s_k, a_k, a_{k+1}) = C_{\max}^{(k)} - C_{\max}^{(k+1)}.$$

Trong cả 2 cách hiểu, việc tối đa hóa phần thưởng tích lũy tương đương với việc tối thiểu hóa thời gian chờ.

On other hand, other works have aimed to minimize makespan by minimizing a highly correlated metric. E.g., Park et al. (2021b) proposes a "waiting job reward function", which defines reward at $R(s_k, a_k, s_{k+1})$ as negative number of jobs waiting at step k.

– Mặt khác, các nghiên cứu khác đã hướng đến việc giảm thiểu makespan bằng cách giảm thiểu 1 số liệu có tương quan cao. Ví dụ, Park & cộng sự (2021b) đề xuất 1 "hàm phần thưởng công việc đang chờ", trong đó phần thưởng tại $R(s_k, a_k, s_{k+1})$ là số lượng công việc đang chờ âm tại bước k.

Another alternative: define it based on "scheduled area" or idle time (Tassel et al., 2021). After each action, difference between duration of scheduled operations & introduced idle times is computed. Thus, scheduled-area-based reward can be defined as

$$R(s_k, a_k, s_{k+1}) = p_{ij} - \sum_{M \in \mathcal{M}} \texttt{empty}_M(s_k, s_{k+1})$$

where p_{ij} is processing time of operation dispatched with action a_k, & $\texttt{empty}_M(s_k, s_{k+1})$ is a function that returns idle time of machine M while transitioning from s_k to s_{k+1}.

– 1 phương án khác: định nghĩa dựa trên "khu vực đã lên lịch" hoặc thời gian nhàn rỗi (Tassel & cộng sự, 2021). Sau mỗi hành động, sự khác biệt giữa thời lượng của các hoạt động đã lên lịch & thời gian nhàn rỗi được đưa vào sẽ được tính toán. Do đó, phần thưởng dựa trên khu vực đã lên lịch có thể được định nghĩa là

$$R(s_k, a_k, s_{k+1}) = p_{ij} - \sum_{M \in \mathcal{M}} \texttt{empty}_M(s_k, s_{k+1})$$

trong đó p_{ij} là thời gian xử lý của hoạt động được phân bổ với hành động a_k, & $\texttt{empty}_M(s_k, s_{k+1})$ là 1 hàm trả về thời gian nhàn rỗi của máy M khi chuyển từ s_k sang s_{k+1}.

- 5.4. Reinforcement Learning Environments. Some open-source RL environments to model JSSP already exist. However, they all force users to use a subset of abovementioned possibilities. E.g., environment presented by Tassel et al. (2021) was not designed to support graph representations & necessary customization. State is represented by a matrix where each row corresponds to a job & columns represent scaled features like remaining processing time, completion percentage, machine availability time, & idle times. Definition of available operations to schedule is also fixed. Legal actions are masked based on job/machine availability & completion status. Environment incorporates heuristics like "non-final prioritization" (prioritizing non-final operations) & rules to restrict reservation of operations. Moreover, only reward function supported is scheduled-area-based one described in prev sect. Another limitation derived from not using graphs: a different model needs to be created for each instance size.

  – Môi trường Học Tăng cường. 1 số môi trường RL nguồn mở để mô hình hóa JSSP đã tồn tại. Tuy nhiên, tất cả chúng đều buộc người dùng phải sử dụng 1 tập hợp con các khả năng đã đề cập ở trên. Ví dụ: môi trường do Tassel & cộng sự (2021) trình bày không được thiết kế để hỗ trợ biểu diễn đồ thị & tùy chỉnh cần thiết. Trạng thái được biểu diễn bằng 1 ma trận, trong đó mỗi hàng tương ứng với 1 công việc & các cột biểu diễn các tính năng được chia tỷ lệ như thời gian xử lý còn lại, tỷ lệ hoàn thành, thời gian khả dụng của máy & thời gian nhàn rỗi. Định nghĩa về các hoạt động khả dụng để lên lịch cũng được cố định. Các hành động pháp lý được che dấu dựa trên công việc /tính khả dụng của máy & trạng thái hoàn thành. Môi trường kết hợp các phương pháp tìm kiếm như "ưu tiên không phải cuối cùng" (ưu tiên các hoạt động không phải cuối cùng) & các quy tắc để hạn chế việc đặt trước các hoạt động. Hơn nữa, chỉ có hàm phần thưởng được hỗ trợ là hàm dựa trên vùng đã lên lịch được mô tả trong phần trước. 1 hạn chế khác xuất phát từ việc không sử dụng đồ thị: cần tạo 1 mô hình khác nhau cho mỗi kích thước phiên bản.

  To best of our knowledge, most customizable open-source RL environment is being developed by Infantes et al. (2024). Their implementation allows customizing reward function & supports GNNs. However, some of critical components we identified previously are non-customizable, including node features, graph representation, & definition of available actions. Additionally, modifying current implementation components is hard due to lack of documentation.

  – Theo hiểu biết của chúng tôi, hầu hết các môi trường RL nguồn mở có thể tùy chỉnh đang được Infantes & cộng sự (2024) phát triển. Việc triển khai của họ cho phép tùy chỉnh hàm thưởng & hỗ trợ GNN. Tuy nhiên, 1 số thành phần quan trọng mà chúng tôi đã xác định trước đây không thể tùy chỉnh, bao gồm các tính năng nút, biểu diễn đồ thị, & định nghĩa các hành động khả dụng. Ngoài ra, việc sửa đổi các thành phần triển khai hiện tại rất khó khăn do thiếu tài liệu hướng dẫn.

  Other works have also developed RL environments to support their experiments Zhang et al. (2020); Song et al. (2023). Nevertheless, they only support their design choices (e.g., a specific set of node features or graph representation) as well. Similarly, they lack modularity & documentation to customize environment further. Other authors have not made their code public.

  – Các công trình khác cũng đã phát triển môi trường RL để hỗ trợ các thí nghiệm của họ (Zhang & cộng sự, 2020; Song & cộng sự, 2023). Tuy nhiên, chúng cũng chỉ hỗ trợ các lựa chọn thiết kế của riêng chúng (ví dụ: 1 tập hợp các đặc điểm nút hoặc biểu diễn đồ thị cụ thể). Tương tự, chúng thiếu tính mô-đun & tài liệu hướng dẫn để tùy chỉnh môi trường hơn nữa. Các tác giả khác chưa công khai mã của họ.

- 5.5. Summary of GNN-Based Dispatchers. Summarizing the approaches mentioned in the following sections helps to have a better perspective for Chap. 8, in which we train our GNN-based dispatcher using JobShopLib's components. Table 5.2: Comparison of GNN-based dispatchers shows this analysis.

  – Tóm tắt các phương pháp được đề cập trong các phần sau giúp có góc nhìn tốt hơn cho Chương 8, trong đó chúng tôi đào tạo bộ điều phối dựa trên GNN bằng các thành phần của JobShopLib. Bảng 5.2: So sánh các bộ điều phối dựa trên GNN thể hiện phân tích này.

II. CONTRIBUTION & RESULTS.

- 6. Introducing `JobShopLib`. A flexible RL environment for JSSP should implement many functionalities. E.g., it should be able to generate instances randomly, calculate active or semi-active schedule sequentially, update graph, or compute node features for each step. To support all these features in a reusable way, have developed `JobShopLib`. These tasks can be helpful on their

own. Therefore, creating this library has advantage of unlocking many more applications beyond solving problem with RL. E.g., this project shows how these components can be easily used to generate an IL dataset. In this chap, introduce some of library's core components & its philosophy. Specifically, all classes & functions necessary to solve JSSP with PDRs are described following a 1st-principle approach.

– 1 môi trường RL linh hoạt cho JSSP nên triển khai nhiều chức năng. Ví dụ: nó phải có khả năng tạo ngẫu nhiên các thể hiện, tính toán lịch trình hoạt động hoặc bán hoạt động tuần tự, cập nhật đồ thị hoặc tính toán các đặc trưng nút cho mỗi bước. Để hỗ trợ tất cả các tính năng này theo cách có thể tái sử dụng, chúng tôi đã phát triển `JobShopLib`. Các tác vụ này có thể hữu ích riêng lẻ. Do đó, việc tạo ra thư viện này có lợi thế là mở ra nhiều ứng dụng hơn ngoài việc giải quyết vấn đề bằng RL. Ví dụ: dự án này cho thấy cách các thành phần này có thể dễ dàng được sử dụng để tạo tập dữ liệu IL. Trong chương này, chúng tôi sẽ giới thiệu 1 số thành phần cốt lõi của thư viện & triết lý của nó. Cụ thể, tất cả các lớp & hàm cần thiết để giải quyết JSSP với PDR được mô tả theo phương pháp nguyên lý thứ nhất.

Begin by examining foundational data structures `JobShopInstance, Operation, Schedule, ScheduledOperation` that form backbone of library. Next, discuss file formats & benchmark instances, showing how problem instances are stored, retrieved, & shared. Then, explain how to generate random instances. These features can be used to evaluate & compare different dispatchers, including PDRs.

– Bắt đầu bằng việc xem xét các cấu trúc dữ liệu nền tảng `JobShopInstance, Operation, Schedule, ScheduledOperation` tạo nên xương sống của thư viện. Tiếp theo, thảo luận về các định dạng tệp & các phiên bản chuẩn, cho thấy cách các phiên bản có vấn đề được lưu trữ, truy xuất, & chia sẻ. Sau đó, giải thích cách tạo các phiên bản ngẫu nhiên. Các tính năng này có thể được sử dụng để đánh giá & so sánh các bộ điều phối khác nhau, bao gồm cả PDR.

However, still need to handle core scheduling logic – which is encapsulated in `Dispatcher` class. It determines operation start times while respecting all constraints. This class also allows you to use different ready operation filters that define available operations at each decision point. Subsequently, explore Observer pattern implementation that allows components to react to dispatching even without tight coupling.

– Tuy nhiên, vẫn cần xử lý logic lập lịch cốt lõi – được đóng gói trong lớp `Dispatcher`. Lớp này xác định thời gian bắt đầu hoạt động trong khi vẫn tuân thủ tất cả các ràng buộc. Lớp này cũng cho phép bạn sử dụng các bộ lọc hoạt động sẵn sàng khác nhau để xác định các hoạt động khả dụng tại mỗi điểm quyết định. Sau đó, hãy khám phá việc triển khai mẫu Observer cho phép các thành phần phản ứng với việc điều phối ngay cả khi không có liên kết chặt chẽ.

Finally, examine implementation of PDRs & how they interface with `Dispatcher` through `DispatchingRuleSolver` to create complete schedules. Throughout chap, emphasize design choices that favor code reusability & extensibility.

– Cuối cùng, hãy xem xét việc triển khai PDR & cách chúng tương tác với `Dispatcher` thông qua `DispatchingRuleSolver` để tạo ra các lịch trình hoàn chỉnh. Trong suốt chương này, hãy nhấn mạnh các lựa chọn thiết kế ưu tiên khả năng tái sử dụng mã & khả năng mở rộng.

- 6.1. Main Data Structures. In his famous book Clean Code (Martin, 2008), Robert C. Martin distinguishes between "data structures" & "objects" [This use of "object" should not be confused with Python's definition, which defines everything, including data structures or even integers, as objects.] as fundamental but opposite abstractions. Data structures expose their data & have no meaningful behavior – they are simply containers that hold information, with public variables & few (or no) methods. Objects, on other hand, hide their data behind abstractions & expose methods that operate on that data, without revealing how data is stored or manipulated internally. This difference creates a fundamental dichotomy: Data structures make it easy to add new functions without changing existing classes. In contrast, object make it easy to add new classes without changing existing methods.

  – Main Data Structures. Trong cuốn sách nổi tiếng Clean Code (Martin, 2008), Robert C. Martin phân biệt giữa "cấu trúc dữ liệu" & "đối tượng" [Không nên nhầm lẫn cách sử dụng "đối tượng" này với định nghĩa của Python, định nghĩa mọi thứ, bao gồm cả cấu trúc dữ liệu hoặc thậm chí số nguyên, là đối tượng.] là các phép trừu tượng cơ bản nhưng ngược lại. Cấu trúc dữ liệu phơi bày dữ liệu của chúng & không có hành vi có ý nghĩa – chúng chỉ đơn giản là các thùng chứa thông tin, với các biến công khai & ít (hoặc không có) phương thức. Mặt khác, đối tượng ẩn dữ liệu của chúng đằng sau các phép trừu tượng & phơi bày các phương thức hoạt động trên dữ liệu đó, mà không tiết lộ cách dữ liệu được lưu trữ hoặc thao tác bên trong. Sự khác biệt này tạo ra 1 sự phân đôi cơ bản: Cấu trúc dữ liệu giúp dễ dàng thêm các hàm mới mà không cần thay đổi các lớp hiện có. Ngược lại, đối tượng giúp dễ dàng thêm các lớp mới mà không cần thay đổi các phương thức hiện có.

  In research, not just interested in finding a solution for a particular JSSP (object approach), but in testing different algorithms & comparing them. For this reason, in JobShopLib, have opted to favor creation of new functions by developing core data structures that are agnostic to solving method. These main data structures are:

  1. `JobShopInstance` serves as central data structure for representing JSSP instances. It maintains a list of jobs, each composed of an ordered sequence (another list) of operations, along with a name & optional metadata. Class provides numerous properties that derive useful information from instance, e.g. matrix representations of operation durations & machine assignments, machine loads, & job durations. These properties are cached for performance when they require expensive computations.
  2. `Operation` is atom of library. Each operation represents a task with a specific duration that must be processed on 1 or more machines. I.e., class supports both classical job shop problems, where each operations can only be processed on 1 specific machine (focus of this project), & flexible problems, where operations may have multiple possible machines. Operations

maintain references to their containing job through attributes like `jod_id, position_in_job, operation_id`, which are typically set by `JobShopInstance` class after initialization.

3. Schedule represents a solution (complete or partial) for a particular JSSP instance. It stores a list of lists of `ScheduledOperation` objects, where each list represents sequence of operations assigned to a specific machine. Class provides methods to calculate makespan, check solution completeness, & add new `ScheduledOperations` to schedule.
4. ScheduledOperation encapsulates assignment of an `Operation` to a specific machine at a particular start time. It maintains references to original operation while adding scheduling information (i.e., start time & assigned machine ID). Class provides properties to derive useful information, e.g. end time, job ID, & "position in job", from underlying operation.

Note, e.g.: `Schedule` class is agnostic to logic used for setting start time of a `ScheduledOperation`. Its `add` method receives an already initialized `ScheduledOperation`. This design choice decouples schedule from scheduling logic.

– Trong nghiên cứu, chúng tôi không chỉ quan tâm đến việc tìm ra giải pháp cho 1 JSSP (phương pháp tiếp cận đối tượng) cụ thể, mà còn thử nghiệm các thuật toán khác nhau & so sánh chúng. Vì lý do này, JobShopLib đã chọn ưu tiên việc tạo các hàm mới bằng cách phát triển các cấu trúc dữ liệu cốt lõi không phụ thuộc vào phương pháp giải quyết. Các cấu trúc dữ liệu chính này là:

1. `JobShopInstance` đóng vai trò là cấu trúc dữ liệu trung tâm để biểu diễn các thể hiện JSSP. Nó duy trì 1 danh sách các công việc, mỗi công việc bao gồm 1 chuỗi các thao tác được sắp xếp theo thứ tự (1 danh sách khác), cùng với 1 tên & siêu dữ liệu tùy chọn. Lớp cung cấp nhiều thuộc tính lấy thông tin hữu ích từ thể hiện, ví dụ: biểu diễn ma trận thời lượng thao tác & phân công máy, tải máy, & thời lượng công việc. Các thuộc tính này được lưu trữ đệm để tăng hiệu suất khi chúng yêu cầu các phép tính tốn kém.
2. `Operation` là 1 phần tử của thư viện. Mỗi thao tác đại diện cho 1 tác vụ có thời lượng cụ thể phải được xử lý trên 1 hoặc nhiều máy. Tức là, lớp này hỗ trợ cả các bài toán job shop cổ điển, trong đó mỗi thao tác chỉ có thể được xử lý trên 1 máy cụ thể (trọng tâm của dự án này), & các bài toán linh hoạt, trong đó các thao tác có thể có nhiều máy khả thi. Các thao tác duy trì tham chiếu đến công việc chứa chúng thông qua các thuộc tính như `jod_id, position_in_job, operation_id`, thường được thiết lập bởi lớp `JobShopInstance` sau khi khởi tạo.
3. Schedule đại diện cho 1 giải pháp (toàn bộ hoặc 1 phần) cho 1 thể hiện JSSP cụ thể. Nó lưu trữ 1 danh sách các đối tượng `ScheduledOperation`, trong đó mỗi danh sách đại diện cho chuỗi các thao tác được gán cho 1 máy cụ thể. Lớp cung cấp các phương thức để tính toán khoảng thời gian hoàn thành, kiểm tra tính đầy đủ của giải pháp & thêm `ScheduledOperations` mới vào lịch trình.
4. ScheduledOperation đóng gói việc gán 1 `Thao tác` cho 1 máy cụ thể tại 1 thời điểm bắt đầu cụ thể. Nó duy trì các tham chiếu đến thao tác ban đầu đồng thời thêm thông tin lập lịch (tức là thời gian bắt đầu & ID máy được gán). Lớp cung cấp các thuộc tính để lấy thông tin hữu ích, ví dụ: thời gian kết thúc, ID công việc, & "vị trí trong công việc", từ thao tác cơ bản.

Lưu ý, ví dụ: lớp `Schedule` không phụ thuộc vào logic được sử dụng để thiết lập thời gian bắt đầu của `ScheduledOperation`. Phương thức `add` của lớp này nhận 1 `ScheduledOperation` đã được khởi tạo. Lựa chọn thiết kế này tách rời lịch trình khỏi logic lập lịch.

Having this logic decoupled also facilitates extending these data structures to support more realistic definitions of JSSP. E.g., library already supports flexible JSSP by allowing an `Operation` to be completed on multiple machines. Other features, e.g. setup times for machines or due dates, can be incorporated by extending original `Operation` through inheritance or by adding this information into `JobShopInstance`'s `metadata` attribute.

– Việc tách rời logic này cũng giúp mở rộng các cấu trúc dữ liệu này để hỗ trợ các định nghĩa thực tế hơn về JSSP. Ví dụ: thư viện đã hỗ trợ JSSP linh hoạt bằng cách cho phép 1 `Operation` được hoàn thành trên nhiều máy. Các tính năng khác, ví dụ như thời gian thiết lập cho máy hoặc ngày đến hạn, có thể được tích hợp bằng cách mở rộng `Operation` ban đầu thông qua kế thừa hoặc bằng cách thêm thông tin này vào thuộc tính `metadata` của `JobShopInstance`.

Another features of these data structures is use of `__slots__` for storing their attributes. When a class defines `__slots__`, it explicitly declares which attributes instances of that class may possess, replacing default dictionary-based attribute storage mechanism `__dict__` with a more efficient array-like structure. This approach achieves memory savings & reduces attribute access latency by avoiding overhead associated with hash table implementations required by Python's dictionaries. This reduction in latency is particularly noticeable when solvers need to access these classes' attributes frequently. `JobShopInstance` class is only class that uses `__dict__` attribute. Reason: it needs this dictionary representation to store its cached attributes.

– 1 tính năng khác của các cấu trúc dữ liệu này là sử dụng `__slots__` để lưu trữ các thuộc tính của chúng. Khi 1 lớp định nghĩa `__slots__`, nó sẽ khai báo rõ ràng các thuộc tính mà các thể hiện của lớp đó có thể sở hữu, thay thế cơ chế lưu trữ thuộc tính mặc định dựa trên từ điển `__dict__` bằng 1 cấu trúc dạng mảng hiệu quả hơn. Cách tiếp cận này tiết kiệm bộ nhớ & giảm độ trễ truy cập thuộc tính bằng cách tránh chi phí liên quan đến việc triển khai bảng băm mà từ điển Python yêu cầu. Độ trễ giảm này đặc biệt đáng chú ý khi trình giải cần truy cập thường xuyên vào các thuộc tính của các lớp này. Lớp `JobShopInstance` là lớp duy nhất sử dụng thuộc tính `__dict__`. Lý do: nó cần biểu diễn từ điển này để lưu trữ các thuộc tính được lưu trong bộ nhớ đệm.

- 6.2. Reading & Saving JSSP Instances & Solutions. Comparing performance of an algorithm to solve JSSP with other methods requires evaluating them with same set of JSSP instances. This set then constitutes a benchmark. To create these benchmarks, need a way to save a JSSP instance to a file that other practitioners can read.

– Đọc & Lưu các Phiên bản JSSP & Giải pháp. Việc so sánh hiệu suất của 1 thuật toán để giải quyết JSSP với các phương pháp khác đòi hỏi phải đánh giá chúng với cùng 1 tập hợp các phiên bản JSSP. Tập hợp này sau đó tạo thành 1 chuẩn mực. Để tạo ra các chuẩn mực này, cần có cách lưu 1 phiên bản JSSP vào 1 tệp mà những người thực hành khác có thể đọc được.

* 6.2.1. Taillard Format. Classic JSSP instances are typically stored using Taillard file format, which is named after Éric Taillard, who introduced benchmark instances for various scheduling problems in early 1990s (Taillard, 1993). General format consists of a header sect & a matrix representation of machines & processing times. Header typically includes number of machines & jobs. E.g., JSSP instance introduced in introductory chap (Ex1) would be represented as:

```
# We can also add comments using "#"
3 3
0 2 1 2 2 2
0 1 1 1 2 1
0 2 2 3 1 3
```

In this case, `3 3` denotes: there are 3 jobs & 3 machines in scheduling problem. Each subsequent line represents a job, consisting of a sequence of operations, where each operation is defined by a machine-processing time pair. 1st number in each pair indicates machine ID (index from 0), & 2nd number represents processing time required for that operation. E.g., sequence `0 2 1 2 2 2` can be interpreted as follows: 1st operation uses machine 0 (cutting machine) for 2 hours, 2nd operation requires machine 1 (sanding machine) for 2 hours, & 3rd operation needs machine 2 (assembly station) for 2 hours.
– Định dạng Taillard. Các phiên bản JSSP cổ điển thường được lưu trữ bằng định dạng tệp Taillard, được đặt theo tên của Éric Taillard, người đã giới thiệu các phiên bản chuẩn cho nhiều bài toán lập lịch khác nhau vào đầu những năm 1990 (Taillard, 1993). Định dạng chung bao gồm 1 phần tiêu đề & 1 ma trận biểu diễn các máy & thời gian xử lý. Phần tiêu đề thường bao gồm số lượng máy & công việc. Ví dụ: phiên bản JSSP được giới thiệu trong chương giới thiệu (Ví dụ 1) sẽ được biểu diễn như sau:

```
# Chúng ta cũng có thể thêm chú thích bằng cách sử dụng "#"
3 3
0 2 1 2 2 2
0 1 1 1 2 1
0 2 2 3 1 3
```

Trong trường hợp này, `3 3` biểu thị: có 3 công việc & 3 máy trong bài toán lập lịch. Mỗi dòng tiếp theo biểu diễn 1 công việc, bao gồm 1 chuỗi các thao tác, trong đó mỗi thao tác được xác định bởi 1 cặp thời gian xử lý của máy. Số thứ nhất trong mỗi cặp biểu thị ID máy (chỉ số bắt đầu từ 0), & số thứ hai biểu thị thời gian xử lý cần thiết cho thao tác đó. Ví dụ: chuỗi `0 2 1 2 2 2` có thể được diễn giải như sau: Thao tác thứ nhất sử dụng máy 0 (máy cắt) trong 2 giờ, thao tác thứ hai cần máy 1 (máy chà nhám) trong 2 giờ, & thao tác thứ ba cần máy 2 (trạm lắp ráp) trong 2 giờ.
`JopShopLib`'s `JobShopInstance` class provides a class method `from_taillard_file` to load an instance from this type of file.

* 6.2.2. JobShopLib's JSON-Based Format. Typically, benchmark instances are accompanied by metadata, e.g. lower & upper bounds or optimum makespan if known. Parsing this information from header format of a Taillard file can be cumbersome because there are no standard "keys" for this type of information. Additionally, while Taillard's format allows us to store JSSP instances, it does not support storing solutions or JSSP variants.
– Thông thường, các phiên bản chuẩn được kèm theo siêu dữ liệu, ví dụ: giới hạn dưới & trên hoặc khoảng thời gian tối ưu (nếu biết). Việc phân tích thông tin này từ định dạng tiêu đề của tệp Taillard có thể phức tạp vì không có "khóa" chuẩn nào cho loại thông tin này. Ngoài ra, mặc dù định dạng của Taillard cho phép chúng tôi lưu trữ các phiên bản JSSP, nhưng nó không hỗ trợ lưu trữ các giải pháp hoặc biến thể JSSP.
p. 60+++

  - 6.3. Generating Random Instances.
  - 6.4. `Dispatcher` Class.
  - 6.5. Dispatching Rules.

- 7. Reinforcement Learning Environment.
- 8. Experiments & Results.
- 9. Ethical, Environmental, & Social Aspects.
- Conclusions & Future Directions.

4.2 Igor G. Smit et al. GNNs for JSSPs: A Survey

[21 citations]

- Abstract. JSSPs represent a critical & challenging class of combinatorial optimization problems. Recent years have witnessed a rapid increase in application of GNNs to solve JSSPs, albeit lacking a systematic survey of relevant literature. This paper aims to thoroughly review prevailing GNN methods for different types of JSSPs & closely related flow-shop scheduling problems (FSPs), especially those leveraging deep reinforcement learning (DRL). Begin by presenting graph representations of various JSSPs, followed by an introduction to most commonly used GNN architectures. Then review current GNN-based methods

for each problem type, highlighting key technical elements e.g. graph representations, GNN architectures, GNN tasks, & training algorithms. Finally, summarize & analyze advantages & limitations of GNNs in solving JSSPs & provide potential future research opportunities. Hope this survey can motivate & inspire innovative approaches for more powerful GNN-based approaches in tackling JSSPs & other scheduling problems.

– JSSP đại diện cho 1 lớp bài toán tối ưu hóa tổ hợp quan trọng & đầy thách thức. Những năm gần đây đã chứng kiến sự gia tăng nhanh chóng trong việc ứng dụng GNN để giải quyết JSSP, mặc dù thiếu 1 cuộc khảo sát có hệ thống về các tài liệu liên quan. Bài báo này nhằm mục đích xem xét kỹ lưỡng các phương pháp GNN phổ biến cho các loại JSSP khác nhau & các vấn đề lập lịch flow-shop (FSP) có liên quan chặt chẽ, đặc biệt là các phương pháp tận dụng học tăng cường sâu (DRL). Bắt đầu bằng cách trình bày các biểu diễn đồ thị của các JSSP khác nhau, tiếp theo là phần giới thiệu về các kiến trúc GNN được sử dụng phổ biến nhất. Sau đó, xem xét các phương pháp dựa trên GNN hiện tại cho từng loại bài toán, làm nổi bật các yếu tố kỹ thuật chính, ví dụ như biểu diễn đồ thị, kiến trúc GNN, tác vụ GNN, & các thuật toán đào tạo. Cuối cùng, tóm tắt & phân tích các ưu điểm & hạn chế của GNN trong việc giải quyết JSSP & cung cấp các cơ hội nghiên cứu tiềm năng trong tương lai. Hy vọng cuộc khảo sát này có thể thúc đẩy & truyền cảm hứng cho các phương pháp tiếp cận sáng tạo để có các phương pháp tiếp cận dựa trên GNN mạnh mẽ hơn trong việc giải quyết JSSP & các vấn đề lập lịch khác.

- **Keywords.** GNN, DRL, job shop scheduling, flow-shop scheduling, combinatorial optimization.

- 1. Introduction. JSSPs are a significant class of NP-hard scheduling problems. While being challenging, JSSPs are broadly investigated in computer science & operations research, with various applications in manufacturing (Gupta & Sivakumar, 2006; Zhang et al., 2019; Wang et al., 2021), healthcare (Pham & Klinkert, 2008; Sarfaraj et al., 2021; Lachtar & Driss, 2023), & supply chain management (Liu & Kozan, 2016; Liao & Lin, 2019; Cai et al., 2023). Traditional methods for solving JSSPs are mainly exact & heuristic algorithms (Jain & Meeran, 1999; Chaudhry & Khan, 2016; Xie et al., 2019; Baptiste et al., 2001; Xiong et al., 2022). Exact algorithms are designed to find optimal solutions but can suffer from inefficiency & limited scalability. In contrast, heuristic algorithms are more efficient & scalable, yet they may compromise quality of found solution. These 2 classes of approaches rely heavily on domain knowledge & modeling efforts to identify key parameters, important constraints between parameters, & objective functions.

  – JSSP là 1 lớp quan trọng của các bài toán lập lịch NP-khó. Mặc dù đầy thách thức, JSSP được nghiên cứu rộng rãi trong khoa học máy tính & nghiên cứu vận hành, với nhiều ứng dụng khác nhau trong sản xuất (Gupta & Sivakumar, 2006; Zhang & cộng sự, 2019; Wang & cộng sự, 2021), chăm sóc sức khỏe (Pham & Klinkert, 2008; Sarfaraj & cộng sự, 2021; Lachtar & Driss, 2023), & quản lý chuỗi cung ứng (Liu & Kozan, 2016; Liao & Lin, 2019; Cai & cộng sự, 2023). Các phương pháp truyền thống để giải JSSP chủ yếu là các thuật toán chính xác & heuristic (Jain & Meeran, 1999; Chaudhry & Khan, 2016; Xie & cộng sự, 2019; Baptiste & cộng sự, 2001; Xiong & cộng sự, 2022). Các thuật toán chính xác được thiết kế để tìm ra các giải pháp tối ưu nhưng có thể gặp phải tình trạng kém hiệu quả & khả năng mở rộng hạn chế. Ngược lại, các thuật toán heuristic hiệu quả hơn & khả năng mở rộng, nhưng chúng có thể làm giảm chất lượng của giải pháp tìm được. Hai lớp tiếp cận này phụ thuộc rất nhiều vào kiến thức chuyên môn & nỗ lực mô hình hóa để xác định các tham số chính, các ràng buộc quan trọng giữa các tham số, & hàm mục tiêu.

  In recent years, another class of approach, i.e., ML-based methods, has gained increasing attention from researchers. In particular, GNNs have been extensively applied to solve JSSPs & have demonstrated promising results. Rationale behind such applications is 2fold:

  1. Real-world JSSPs need to be solved daily, with massive data of historical instances available. These data can be leveraged for training neural networks, enabling automatic learning of heuristics for efficiently solving a set of problem instances.
  2. JSSPs can be suitably represented by graphs, with relationships between jobs & machines depicted by graph topology. GNNs are adept at capturing favorable representations of JSSP instances, which aid in learning effective heuristics for solving them.

  Despite numerous GNN-based methods being proposed for JSSPs, there is a notable absence of a comprehensive review to gauge real process made by these advanced DL methods. This paper aims to fill this gap by reviewing prevailing GNN-based methods for various types of JSSPs, including basic JSSP, flexible JSSP, dynamic JSSP, & distributed JSSP, as well as similar problems like flow-shop scheduling problem (FSP), hybrid FSP, & permutation FSP.

  – Trong những năm gần đây, 1 hướng tiếp cận khác, tức là các phương pháp dựa trên ML, đã ngày càng thu hút sự chú ý của các nhà nghiên cứu. Cụ thể, GNN đã được ứng dụng rộng rãi để giải quyết JSSP & đã cho thấy những kết quả đầy hứa hẹn. Lý do đằng sau những ứng dụng này là:

  1. JSSP thực tế cần được giải quyết hàng ngày, với lượng dữ liệu khổng lồ về các trường hợp lịch sử có sẵn. Những dữ liệu này có thể được tận dụng để huấn luyện mạng nơ-ron, cho phép tự động học các phương pháp tìm kiếm để giải quyết hiệu quả 1 tập hợp các trường hợp bài toán.
  2. JSSP có thể được biểu diễn phù hợp bằng đồ thị, với mối quan hệ giữa các công việc & máy móc được mô tả bằng tôpô đồ thị. GNN rất giỏi trong việc nắm bắt các biểu diễn thuận lợi của các trường hợp JSSP, hỗ trợ việc học các phương pháp tìm kiếm hiệu quả để giải quyết chúng.

  Mặc dù có nhiều phương pháp dựa trên GNN được đề xuất cho JSSP, nhưng vẫn chưa có 1 đánh giá toàn diện nào để đánh giá quy trình thực tế được thực hiện bởi các phương pháp DL tiên tiến này. Bài báo này nhằm mục đích lấp đầy khoảng trống này bằng cách xem xét các phương pháp dựa trên GNN hiện hành cho nhiều loại JSSP khác nhau, bao gồm JSSP cơ bản,

JSSP linh hoạt, JSSP động, & JSSP phân tán, cũng như các vấn đề tương tự như vấn đề lập lịch flow-shop (FSP), FSP lai, & FSP hoán vị.

Although several review papers on application of ML in solving JSSPs exist, a comprehensive review specifically focused on advanced GNN-basd methods for JSSPs is still lacking. Çaliş & Bulkan (2015) & Zhang et al. (2019) review basic ML-based methods combined with traditional heuristic algorithms for JSSPs, overlooking recent advancements of DL methods. Cunha et al. (2020) provide a conceptual framework for applying DRL to solve JSSPs without delving into further discussions on natural architectures & training algorithms that are applicable to JSSPs. Li et al. (2021) present a survey from a bibliometric perspective, offering general statistics on ML-based solutions for JSSPs without a comprehensive overview of technical details. More recently, Zhang et al. (2023b) discuss mixed genetic programming & ML techniques for JSSPs, & Zhang et al. (2023a) review DL techniques for vehicle routing, job shop scheduling, & bin packing problems from a manufacturing perspective. Otala et al. (2021) summarize traditional applications of graphs in shop scheduling problems but ignore power of GNNs to automate policy learning. These reviews provide limited descriptions for each problem type, often omitting detailed technical insights, particularly on how GNNs are structured & utilized in each method. Instead, in this paper, focus on reviewing GNN-based methods for solving different types of JSSPs. Rationale for such a survey is multifaceted. 1stly, GNNs have been most researched models in realm of DL for JSSPs, as instances of JSSPs are characterized by clear graph topologies. 2ndly, GNN-based methods for JSSPs can facilitate development of DL techniques for other scheduling problems. E.g.,GNNs with DRL training algorithms for JSSPs can expedite development of similar methods for machine scheduling problems & task scheduling problems, as highlighted in Kayhan & Yildiz (2023), Liu et al. (2024b). Lastly, while surveys on GNN applications in broader combinatorial optimization domains are available (Huang et al., 2019; Peng et al., 2021; Cappart et al., 2023), there is a lack of review on GNN applications in a specific problem e.g. JSSPs. Given substantial recent proposals of GNN-based methods for JSSPs, critical to provide a fine-grained survey of this rapidly evolving domain. This focused review, offering insights into development of GNNs for JSSPs, promises to be more valuable & inspiring than a generic list of literature across entire optimization domain.

– Mặc dù đã có 1 số bài báo đánh giá về ứng dụng ML trong việc giải quyết JSSP, nhưng vẫn còn thiếu 1 đánh giá toàn diện tập trung cụ thể vào các phương pháp GNN-basd tiên tiến cho JSSP. Çaliş & Bulkan (2015) & Zhang & cộng sự (2019) đánh giá các phương pháp cơ bản dựa trên ML kết hợp với các thuật toán tìm kiếm truyền thống cho JSSP, bỏ qua những tiến bộ gần đây của các phương pháp DL. Cunha & cộng sự (2020) cung cấp 1 khuôn khổ khái niệm để áp dụng DRL để giải quyết JSSP mà không đi sâu vào các cuộc thảo luận sâu hơn về kiến trúc tự nhiên & các thuật toán đào tạo có thể áp dụng cho JSSP. Li & cộng sự (2021) trình bày 1 cuộc khảo sát từ góc độ thư mục trắc lượng, cung cấp số liệu thống kê chung về các giải pháp dựa trên ML cho JSSP mà không có tổng quan toàn diện về các chi tiết kỹ thuật. Gần đây hơn, Zhang & cộng sự (2023b) thảo luận về lập trình di truyền hỗn hợp & các kỹ thuật ML cho JSSP, & Zhang & cộng sự (2023a) đánh giá các kỹ thuật DL để định tuyến xe, lập lịch xưởng gia công, & các vấn đề đóng gói thùng từ góc độ sản xuất. Otala & cộng sự (2021) tóm tắt các ứng dụng truyền thống của đồ thị trong các vấn đề lập lịch cửa hàng nhưng bỏ qua sức mạnh của GNN trong việc tự động hóa việc học chính sách. Các bài đánh giá này cung cấp các mô tả hạn chế cho từng loại vấn đề, thường bỏ qua những hiểu biết kỹ thuật chi tiết, đặc biệt là về cách GNN được cấu trúc & sử dụng trong từng phương pháp. Thay vào đó, trong bài báo này, tập trung vào việc xem xét các phương pháp dựa trên GNN để giải quyết các loại JSSP khác nhau. Cơ sở lý luận cho 1 cuộc khảo sát như vậy là nhiều mặt. Thứ nhất, GNN là mô hình được nghiên cứu nhiều nhất trong lĩnh vực DL cho JSSP, vì các trường hợp JSSP được đặc trưng bởi các cấu trúc đồ thị rõ ràng. Thứ hai, các phương pháp dựa trên GNN cho JSSP có thể tạo điều kiện phát triển các kỹ thuật DL cho các vấn đề lập lịch khác. Ví dụ: GNN với các thuật toán đào tạo DRL cho JSSP có thể đẩy nhanh quá trình phát triển các phương pháp tương tự cho các vấn đề lập lịch máy & các vấn đề lập lịch tác vụ, như đã nêu bật trong Kayhan & Yildiz (2023), Liu & cộng sự (2024b). Cuối cùng, mặc dù đã có các khảo sát về ứng dụng GNN trong các lĩnh vực tối ưu hóa tổ hợp rộng hơn (Huang & cộng sự, 2019; Peng & cộng sự, 2021; Cappart & cộng sự, 2023), nhưng vẫn còn thiếu các bài tổng quan về ứng dụng GNN trong 1 bài toán cụ thể, ví dụ như JSSP. Với những đề xuất đáng kể gần đây về các phương pháp dựa trên GNN cho JSSP, việc cung cấp 1 khảo sát chi tiết về lĩnh vực đang phát triển nhanh chóng này là rất quan trọng. Bài tổng quan tập trung này, cung cấp những hiểu biết sâu sắc về sự phát triển của GNN cho JSSP, hứa hẹn sẽ có giá trị & truyền cảm hứng hơn 1 danh sách chung chung về các tài liệu trên toàn bộ lĩnh vực tối ưu hóa.

In this paper, explore different GNNs models for solving JSSPs & categorize research based on training algorithms, namely DRL & non-DRL approaches. This categorization aims to reflect how GNNs can be applied across different training paradigms. Additionally, extend our survey to include several FSPs that bear similarities to JSSPs, with goal of inspiring research community regarding versatile applications of GNNs in solving different types of scheduling problems & highlighting disparity between these applications. In summary, main contributions of this survey paper are outlined as follows:

- Describe typical JSSPs from graph perspective by reviewing graph representations of different types of JSSPs. Building on this foundation, introduce commonly used GNNs for solving JSSPs. This allows us to summarize relationships & differences between graph representations of disparate JSSPs & gain insights into how graph topologies of JSSPs are learned by GNNs.
- Review existing GNN-based methods for solving various types of JSSPs, categorizing them into groups based on training paradigm (i.e., DRL & non-DRL). In our descriptions of each method, explicitly provide critical technical details for GNN applications, e.g. graph representations, GNN types, GNN tasks, & training algorithms. Additionally, extend survey to include similar FSPs, aiming to showcase methodology of applying GNNs to different scheduling problems & reflecting their broader applicability.
- After an extensive review of literature on JSSPs & FSPs, summarize advantages & limitations of current GNN-based methods. From this analysis, propose several promising directions for future research in application of GNNs to JSSPs.

– Trong bài báo này, chúng tôi khám phá các mô hình GNN khác nhau để giải quyết JSSP & phân loại nghiên cứu dựa trên các thuật toán huấn luyện, cụ thể là các phương pháp DRL & không phải DRL. Việc phân loại này nhằm mục đích phản ánh cách GNN có thể được áp dụng trên các mô hình huấn luyện khác nhau. Ngoài ra, chúng tôi mở rộng khảo sát của mình để bao gồm 1 số FSP có điểm tương đồng với JSSP, với mục tiêu truyền cảm hứng cho cộng đồng nghiên cứu về các ứng dụng đa dạng của GNN trong việc giải quyết các loại bài toán lập lịch khác nhau & làm nổi bật sự khác biệt giữa các ứng dụng này. Tóm lại, những đóng góp chính của bài báo khảo sát này được tóm tắt như sau:

- Mô tả các JSSP điển hình từ góc độ đồ thị bằng cách xem xét các biểu diễn đồ thị của các loại JSSP khác nhau. Dựa trên nền tảng này, chúng tôi giới thiệu các GNN thường được sử dụng để giải quyết JSSP. Điều này cho phép chúng tôi tóm tắt các mối quan hệ & sự khác biệt giữa các biểu diễn đồ thị của các JSSP khác nhau & hiểu rõ hơn về cách GNN học được các cấu trúc đồ thị của JSSP.
- Xem xét các phương pháp dựa trên GNN hiện có để giải quyết các loại JSSP khác nhau, phân loại chúng thành các nhóm dựa trên mô hình huấn luyện (ví dụ: DRL & không phải DRL). Trong phần mô tả từng phương pháp, hãy cung cấp rõ ràng các chi tiết kỹ thuật quan trọng cho các ứng dụng GNN, ví dụ: biểu diễn đồ thị, loại GNN, tác vụ GNN, & các thuật toán huấn luyện. Ngoài ra, hãy mở rộng khảo sát để bao gồm các FSP tương tự, nhằm mục đích giới thiệu phương pháp luận áp dụng GNN cho các bài toán lập lịch khác nhau & phản ánh khả năng ứng dụng rộng hơn của chúng.
- Sau khi xem xét kỹ lưỡng các tài liệu về JSSP & FSP, hãy tóm tắt những ưu điểm & hạn chế của các phương pháp dựa trên GNN hiện tại. Từ phân tích này, đề xuất 1 số hướng nghiên cứu đầy hứa hẹn trong tương lai về ứng dụng GNN vào JSSP.

- 2. Problem description. Generally, scheduling problems can be represented by a directed acyclic graph (DAG) consisting of operations (nodes) & precedence constraints (edges), which denote a partial order of operations to be executed (i.e., current operation can only start after precedent operation is finished). Task: run operations on a set of machines without violating precedence constraints & others (e.g., resource limit), if any. Each machine can only run 1 operation at a time. Below, present a detailed description of each problem type.

  – Nhìn chung, các bài toán lập lịch có thể được biểu diễn bằng đồ thị phi chu trình có hướng (DAG) bao gồm các thao tác (nút) & ràng buộc thứ tự ưu tiên (cạnh), biểu thị thứ tự 1 phần của các thao tác cần thực hiện (tức là thao tác hiện tại chỉ có thể bắt đầu sau khi thao tác thứ tự ưu tiên kết thúc). Nhiệm vụ: chạy các thao tác trên 1 tập hợp máy mà không vi phạm các ràng buộc thứ tự ưu tiên & các ràng buộc khác (ví dụ: giới hạn tài nguyên), nếu có. Mỗi máy chỉ có thể chạy 1 thao tác tại 1 thời điểm. Dưới đây là mô tả chi tiết về từng loại bài toán.

  1. **Open-shop Scheduling Problem.** In open-shop scheduling problem (OSP), all jobs must be processed on a set of machines, with objective of minimizing total time to complete all jobs, known as makespan. There are no predefined orders in which jobs must visit machines, & each job must be processed once on each machine.

     – **Bài toán Lập lịch Mở xưởng.** Trong bài toán lập lịch mở xưởng (OSP), tất cả các công việc phải được xử lý trên 1 tập hợp máy, với mục tiêu giảm thiểu tổng thời gian hoàn thành tất cả các công việc, được gọi là makespan. Không có thứ tự công việc nào được xác định trước phải đến từng máy, & mỗi công việc phải được xử lý 1 lần trên mỗi máy.

  2. **Flow-shop Scheduling Problem.** Building upon OSP, flow-shop scheduling problem (FSP) requires: each job is processed on a set of machines, while machine sequence is same & fixed for all jobs. A hybrid flow-shop scheduling problem (HFSP) is a generalization of FSP where each stage can have 1 or more parallel machines. Jobs must still go through each stage in a fixed sequence, but they can be processed on any machine within each stage. For permutation flow-shop scheduling problem (PFSP), not only must all jobs go through machines in same order, but order of jobs must also be same on all machines.

     – **Bài toán Lập lịch Flow-shop.** Dựa trên OSP, bài toán lập lịch flow-shop (FSP) yêu cầu: mỗi công việc được xử lý trên 1 tập hợp máy, trong khi trình tự máy là giống nhau & cố định cho tất cả các công việc. Bài toán lập lịch flow-shop lai (HFSP) là 1 dạng tổng quát của FSP, trong đó mỗi giai đoạn có thể có 1 hoặc nhiều máy song song. Các công việc vẫn phải trải qua từng giai đoạn theo 1 trình tự cố định, nhưng chúng có thể được xử lý trên bất kỳ máy nào trong mỗi giai đoạn. Đối với bài toán lập lịch flow-shop hoán vị (PFSP), không chỉ tất cả các công việc phải đi qua các máy theo cùng 1 thứ tự, mà thứ tự các công việc cũng phải giống nhau trên tất cả các máy.

  3. **Job-shop Scheduling Problem.** Unlike FSPs, each job has a unique sequence of operations that must be processed on specific machines in JSSP. Formally, a standard JSSP instance consists of a set of jobs $\mathcal{J}$ & a set of machines $\mathcal{M}$. Each job $J_i \in \mathcal{J}$ has an operation set containing n_i operations $O_i = \{O_{ij}\}_{j=1}^{n_i}$ that must be processed in a specific order (i.e., precedence constraints), represented as $O_{i1} \to \cdots \to O_{in_i}$. Here, each O_{ij} signifies an operation of J_i with a processing time $p_{ij} \in \mathbb{N}$. Each machine can only process 1 job at a time, & preemption is not allowed. Objective in solving a JSSP instance: determine a start time S_{ij} for each operation O_{ij} s.t. makespan $C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$ is minimized while adhering to all constraints. Size of a JSSP instance is $|\mathcal{J}| \times |\mathcal{M}|$.

     – **Vấn đề lập lịch Job-shop.** Không giống như FSP, mỗi công việc có 1 chuỗi hoạt động duy nhất phải được xử lý trên các máy cụ thể trong JSSP. Về mặt hình thức, 1 phiên bản JSSP chuẩn bao gồm 1 tập hợp các công việc $\mathcal{J}$ & 1 tập hợp các máy $\mathcal{M}$. Mỗi công việc $J_i \in \mathcal{J}$ có 1 tập hợp hoạt động chứa n_i hoạt động $O_i = \{O_{ij}\}_{j=1}^{n_i}$ phải được xử lý theo 1 thứ tự cụ thể (tức là các ràng buộc về thứ tự ưu tiên), được biểu diễn là $O_{i1} \to \cdots \to O_{in_i}$. Ở đây, mỗi O_{ij} biểu thị 1 hoạt động của J_i với thời gian xử lý là $p_{ij} \in \mathbb{N}$. Mỗi máy chỉ có thể xử lý 1 công việc tại 1 thời điểm, & không được phép chiếm dụng trước. Mục tiêu khi giải 1 thể hiện JSSP: xác định thời gian bắt đầu S_{ij} cho mỗi thao tác O_{ij} sao cho khoảng thời gian $C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$ được giảm thiểu trong khi vẫn tuân thủ tất cả các ràng buộc. Kích thước của 1 thể hiện JSSP là $|\mathcal{J}| \times |\mathcal{M}|$.

For JSSPs, most works employ disjunctive graphs (Zhang et al., 2020; Park et al., 2021b; Zhang et al., 2024) or augmented graphs with artificial machine nodes (Park et al., 2021a). Below, introduce mainstream disjunctive graph representation. Formally, a JSSP instance is represented by a disjunctive graph $G = (\mathcal{O}, C, D)$ (Błażewicz et al., 2000). $\mathcal{O} = \{O_{ij}|\forall i, j\} \cup \{O_S, O_T\}$ is set of all operations, where O_S, O_T are dummy operations with 0 processing time, representing start & terminal states. C is a set of directed arcs (conjunctions) denoting precedence constraints among operations within same job, while D is a set of undirected arcs (disjunctions) linking operations necessitating same machine for processing. Consequently, solving a JSSP instance entails determining direction of each disjunction, ensuring resultant graph forms a DAG. Longest path from O_S to O_T in a solution is called *critical path*, whose length is makespan of solution. An example of disjunctive graph & a feasible solution of a JSSP instance is illustrated in Fig. 1: Disjunctive graph representation of JSSPs. Left panel represents a 3 (jobs) $\times$3 (machines) JSSP instance. Black arrows are conjunctive arcs, representing precedence among operations within same job. Dotted lines are disjunctive arcs whose directions are to be assigned. Disjunctive arcs (or operation nodes) with same color require same machine for processing. Right panel represents a feasible solution.

– Đối với JSSP, hầu hết các công trình đều sử dụng đồ thị phân biệt (Zhang & cộng sự, 2020; Park & cộng sự, 2021b; Zhang & cộng sự, 2024) hoặc đồ thị tăng cường với các nút máy nhân tạo (Park & cộng sự, 2021a). Dưới đây, xin giới thiệu biểu diễn đồ thị phân biệt chính thống. Về mặt hình thức, 1 thể hiện JSSP được biểu diễn bằng đồ thị phân biệt $G = (\mathcal{O}, C, D)$ (Błażewicz & cộng sự, 2000). $\mathcal{O} = \{O_{ij}|\forall i, j\} \cup \{O_S, O_T\}$ là tập hợp tất cả các phép toán, trong đó O_S, O_T là các phép toán giả với thời gian xử lý bằng 0, biểu diễn trạng thái bắt đầu & kết thúc. C là 1 tập hợp các cung có hướng (liên kết) biểu thị các ràng buộc về thứ tự ưu tiên giữa các thao tác trong cùng 1 công việc, trong khi D là 1 tập hợp các cung không có hướng (liên kết) liên kết các thao tác cần cùng 1 máy để xử lý. Do đó, việc giải 1 thể hiện JSSP đòi hỏi phải xác định hướng của mỗi liên kết, đảm bảo đồ thị kết quả tạo thành 1 DAG. Đường đi dài nhất từ O_S đến O_T trong 1 giải pháp được gọi là *critical path*, có độ dài bằng khoảng của giải pháp. 1 ví dụ về đồ thị liên kết & 1 giải pháp khả thi của 1 thể hiện JSSP được minh họa trong Hình 1: Biểu diễn đồ thị liên kết của JSSP. Bảng bên trái biểu diễn 1 thể hiện JSSP 3 (công việc) $\times$3 (máy). Các mũi tên màu đen là các cung liên kết, biểu thị thứ tự ưu tiên giữa các thao tác trong cùng 1 công việc. Các đường chấm là các cung liên kết có hướng được chỉ định. Các cung liên kết (hoặc các nút thao tác) có cùng màu yêu cầu cùng 1 máy để xử lý. Bảng bên phải thể hiện 1 giải pháp khả thi.

4. **Distributed Job-shop Scheduling Problem.** Distributed job shop scheduling problem (DiJSSP) is a variant of basic JSSP that incorporates spatial distribution among machines involved. Formally, a DiJSSP instance comprises a set of factories $\mathcal{F} = \{f_i\}_{i=1}^{||\mathcal{F}}$, each equipped with its own machines. Problem involves a collection of jobs that must be processed in 1 factory. In DiJSSP, 2 key decisions should be made: 1stly, selecting & assigning each job to a suitable factory, & 2ndly, scheduling operations of jobs within their assigned factory. Once a job begins processing in a selected factory, it cannot be transferred to another. Primary objective: minimize maximum makespan across all factories. An example of disjunctive graph representation & a feasible solution of a DiJSSP instance is illustrated in Fig. 2: Disjunctive graph representation of Distributed JSSPs. Left panel represents a 4 (jobs) $\times$3 (machines) DiJSSP instance. All jobs should be processed in either f_1 or f_2. Right panel represents a feasible solution.

   – **Bài toán lập lịch phân xưởng công việc phân tán.** Bài toán lập lịch phân xưởng công việc phân tán (DiJSSP) là 1 biến thể của JSSP cơ bản kết hợp phân phối không gian giữa các máy liên quan. Về mặt hình thức, 1 thể hiện DiJSSP bao gồm 1 tập hợp các nhà máy $\mathcal{F} = \{f_i\}_{i=1}^{||\mathcal{F}}$, mỗi nhà máy được trang bị các máy riêng. Bài toán liên quan đến 1 tập hợp các công việc phải được xử lý trong 1 nhà máy. Trong DiJSSP, cần đưa ra 2 quyết định chính: 1. Lựa chọn & chỉ định từng công việc cho 1 nhà máy phù hợp, & 2. Lên lịch hoạt động của các công việc trong nhà máy được chỉ định. Khi 1 công việc bắt đầu được xử lý trong 1 nhà máy đã chọn, nó không thể được chuyển sang nhà máy khác. Mục tiêu chính: giảm thiểu khoảng thời gian xử lý tối đa trên tất cả các nhà máy. 1 ví dụ về biểu diễn đồ thị rời rạc & 1 giải pháp khả thi của 1 thể hiện DiJSSP được minh họa trong Hình 2: Biểu diễn đồ thị rời rạc của JSSP phân tán. Bảng bên trái biểu diễn 1 thể hiện DiJSSP gồm 4 (công việc) $\times$3 (máy). Tất cả công việc nên được xử lý trong f_1 hoặc f_2. Bảng bên phải biểu diễn 1 giải pháp khả thi.

5. **Flexible Job-shop Scheduling Problem.** Different from JSSPs, flexible job shop scheduling problem (FJSSP) allows each operation to be processed on any machine from a subset of available machines capable of performing that operation. Formally, each operation O_{ij} can be processed on any machine M_k from its compatible set $M_k \in \mathcal{M}_{ij} \subseteq \mathcal{M}$ with a processing time p_{ijk}. Each operation could be connected to multiple disjunctive arcs in disjunctive graph representation, & thus, solving FJSSP is equivalent to selecting a disjunctive arc for each operation node & fixing its direction. An example of disjunctive graph & a feasible solution of an FJSSP instance is illustrated in Fig. 3: Disjunctive graph representation of Flexible JSSPs. Left panel represents a 3 (jobs) $\times$3 (machines) FJSSP instance. Each operation can be processed on any machine from a subset of available machines capable of performing that operation. Right panel represents a feasible solution.

   – **Bài toán lập lịch xưởng công việc linh hoạt.** Khác với JSSP, bài toán lập lịch xưởng công việc linh hoạt (FJSSP) cho phép xử lý từng thao tác trên bất kỳ máy nào từ 1 tập hợp con các máy có sẵn có khả năng thực hiện thao tác đó. Về mặt hình thức, mỗi thao tác O_{ij} có thể được xử lý trên bất kỳ máy M_k nào từ tập hợp tương thích $M_k \in \mathcal{M}_{ij} \subseteq \mathcal{M}$ với thời gian xử lý p_{ijk}. Mỗi thao tác có thể được kết nối với nhiều cung rời rạc trong biểu diễn đồ thị rời rạc, & do đó, giải FJSSP tương đương với việc chọn 1 cung rời rạc cho mỗi nút thao tác & cố định hướng của nó. 1 ví dụ về đồ thị rời rạc & 1 giải pháp khả thi của 1 thể hiện FJSSP được minh họa trong Hình 3: Biểu diễn đồ thị rời rạc của JSSP linh hoạt. Bảng bên trái biểu diễn 1 thể hiện FJSSP gồm 3 (công việc) $\times$3 (máy). Mỗi thao tác có thể được xử lý trên bất kỳ máy nào từ 1 tập hợp con các máy có khả năng thực hiện thao tác đó. Bảng bên phải thể hiện 1 giải pháp khả thi.

6. **Dynamic Job-shop Scheduling Problem.** Dynamic job shop scheduling problem (DyJSSP) adds a layer of complexity to traditional JSSP by introducing elements of change & uncertainty that occur over time. Concretely, during execution of

schedule, new jobs may arrive unpredictably, or uncertain events may happen, e.g. machine breakdowns, job cancellations, or variations in job processing times. This requires schedule to be flexible & often necessitates real-time adjustments.

– **Bài toán Lập lịch Xưởng Động.** Bài toán lập lịch Xưởng Động (DyJSSP) làm tăng thêm độ phức tạp cho JSSP truyền thống bằng cách đưa vào các yếu tố thay đổi & bất định xảy ra theo thời gian. Cụ thể, trong quá trình thực hiện lịch, các công việc mới có thể đến 1 cách bất ngờ, hoặc các sự kiện bất định có thể xảy ra, ví dụ như máy móc bị hỏng, công việc bị hủy hoặc thời gian xử lý công việc thay đổi. Điều này đòi hỏi lịch trình phải linh hoạt & thường đòi hỏi phải điều chỉnh theo thời gian thực.

7. **Dynamic Flexible Job-Shop Scheduling Problem.** By incorporating both flexibility in job routing from FJSSP & dynamic changes in production environment from DyJSSP, dynamic flexible job-shop scheduling problem (DyFJSSP) offers a realistic model for optimizing production in complex, dynamic environments. Its ability to respond to real-time changes makes it suitable for modern industries that require high levels of efficiency & adaptability, e.g. automotive assembly lines, semiconductor manufacturing, & custom product fabrication.

   – **Bài toán Lập lịch Xưởng Linh hoạt Động.** Bằng cách kết hợp tính linh hoạt trong định tuyến công việc từ FJSSP & những thay đổi động trong môi trường sản xuất từ DyJSSP, bài toán lập lịch xưởng linh hoạt động (DyFJSSP) cung cấp 1 mô hình thực tế để tối ưu hóa sản xuất trong các môi trường phức tạp & năng động. Khả năng phản hồi với những thay đổi theo thời gian thực giúp nó phù hợp với các ngành công nghiệp hiện đại đòi hỏi hiệu suất cao & khả năng thích ứng, ví dụ như dây chuyền lắp ráp ô tô, sản xuất chất bán dẫn, & chế tạo sản phẩm tùy chỉnh.

In addition to directed acyclic graphs, there is a recent interest in leveraging alternative graph-based techniques, e.g. Petri nets, to represent scheduling problems in a more dynamic & concurrent framework. 1 advantage of Petri nets is their direct interaction with DRL agents. Unlike disjunctive graphs, which require additional methods to extract features for optimization, Petri nets naturally integrate with DRL by allowing token distributions to serve as part of system's observation space. This capability makes Petri nets a promising approach for solving scheduling problems (Lassoued & Schwung, 2024).

– Ngoài đồ thị phi chu trình có hướng, gần đây người ta cũng quan tâm đến việc tận dụng các kỹ thuật dựa trên đồ thị thay thế, ví dụ như mạng Petri, để biểu diễn các bài toán lập lịch trong 1 khuôn khổ đồng thời & động hơn. 1 lợi thế của mạng Petri là tương tác trực tiếp với các tác tử DRL. Không giống như đồ thị phân ly, vốn yêu cầu các phương pháp bổ sung để trích xuất các đặc trưng nhằm tối ưu hóa, mạng Petri tích hợp tự nhiên với DRL bằng cách cho phép phân phối token đóng vai trò là 1 phần của không gian quan sát của hệ thống. Khả năng này khiến mạng Petri trở thành 1 phương pháp đầy hứa hẹn để giải quyết các bài toán lập lịch (Lassoued & Schwung, 2024).

- 3. GNNs. GNNs are a family of deep neural networks that can learn a representation of graph-structured data (Battaglia et al., 2018). Most GNNs are categorized into 4 classes.

  1. **Recurrent Graph Neural Network.** Recurrent GNN aims to learn node representations with recurrent neural architectures. It leverages recurrent connections to capture temporal dependencies within data & utilizes graph-based operations to handle relationships between entities represented as nodes in graph.
  2. **Convolutional Graph Neural Network.** Convolutional GNN generalizes operation of convolution from grid data (e.g., images) to graph data. It involves message passing mechanisms to propagate information between nodes, followed by graph convolution operations to update node representations based on their neighborhood structure. This allows it to effectively capture local & global patterns in graph-structured data. Unlike recurrent GNN iteratively using same graph recurrent layer in updating node representations, it applies different graph convolutional layers to extract high-level node representations effectively.
  3. **Graph Autoencoder.** Graph autoencoder encodes nodes or graphs into a latent representation & reconstructs graph data from encoded information in an unsupervised learning manner. By learning an effective representation in latent space, it can be used for various tasks e.g. graph generation & denoising.
  4. **Spatial–Temporal Graph Neural Network.** Spatial–Temporal GNN aims to learn hidden patterns from spatial–temporal graphs. Generally, it considers spatial dependence & temporal dependence simultaneously by integrating graph convolutions to capture spatial dependence with RNNs or CNNs to model temporal dependence. They are particularly suited for tasks where spatial relationships & temporal dynamics are crucial, e.g. traffic prediction, climate modeling, & human activity recognition. Refer interested readers to Wu et al. (2020) for a comprehensive review regarding GNNs.

– GNN là 1 họ mạng nơ-ron sâu có khả năng học cách biểu diễn dữ liệu có cấu trúc đồ thị (Battaglia & cộng sự, 2018). Hầu hết GNN được phân loại thành 4 lớp.

1. **Mạng nơ-ron đồ thị hồi quy.** GNN hồi quy nhằm mục đích học các biểu diễn nút với kiến trúc nơ-ron hồi quy. Nó tận dụng các kết nối hồi quy để nắm bắt các phụ thuộc thời gian trong dữ liệu & sử dụng các phép toán dựa trên đồ thị để xử lý các mối quan hệ giữa các thực thể được biểu diễn dưới dạng các nút trong đồ thị.
2. **Mạng nơ-ron đồ thị tích chập.** GNN tích chập tổng quát hóa phép toán tích chập từ dữ liệu lưới (ví dụ: hình ảnh) sang dữ liệu đồ thị. Nó bao gồm các cơ chế truyền thông điệp để truyền thông tin giữa các nút, tiếp theo là các phép toán tích chập đồ thị để cập nhật các biểu diễn nút dựa trên cấu trúc lân cận của chúng. Điều này cho phép nó nắm bắt hiệu quả các mẫu cục bộ & toàn cục trong dữ liệu có cấu trúc đồ thị. Không giống như GNN hồi quy lặp đi lặp lại sử dụng cùng 1 lớp hồi quy đồ thị trong việc cập nhật biểu diễn nút, nó áp dụng các lớp tích chập đồ thị khác nhau để trích xuất hiệu quả các biểu diễn nút cấp cao.

Bộ mã hóa tự động đồ thị (Graph Autoencoder). Bộ mã hóa tự động đồ thị mã hóa các nút hoặc đồ thị thành 1 biểu diễn tiềm ẩn & tái tạo dữ liệu đồ thị từ thông tin được mã hóa theo phương pháp học không giám sát. Bằng cách học 1 biểu diễn hiệu quả trong không gian tiềm ẩn, nó có thể được sử dụng cho nhiều tác vụ khác nhau, ví dụ như tạo đồ thị & khử nhiễu.

Mạng nơ-ron đồ thị không gian-thời gian (Spatial–Temporal Graph Neural Network). GNN không gian-thời gian (Spatial-Temporal Graph Neural Network) hướng đến việc học các mẫu ẩn từ đồ thị không gian-thời gian. Nhìn chung, nó xem xét đồng thời sự phụ thuộc không gian & phụ thuộc thời gian bằng cách tích hợp các phép tích chập đồ thị để nắm bắt sự phụ thuộc không gian với RNN hoặc CNN để mô hình hóa sự phụ thuộc thời gian. Chúng đặc biệt phù hợp cho các tác vụ mà mối quan hệ không gian & động lực thời gian là rất quan trọng, ví dụ như dự đoán giao thông, mô hình hóa khí hậu & nhận dạng hoạt động của con người. Giới thiệu Wu & cộng sự, nếu bạn quan tâm, đến Wu & cộng sự. (2020) để có 1 đánh giá toàn diện về GNN.

Due to efficiency, generality, & effectiveness of convolutional GNN, its popularity has been rapidly growing in a wide spectrum of applications, including solving NP-hard combinatorial optimization problems. In this survey, we found convolutional GNNs are often used to solve JSSPs. Formally, a graph is represented as $G = \{V, E\}$, where V, E are sets of nodes & edges, resp. Let $v_i \in V$ denote a nnode, & $e_{ij} \in E$ denote an edge pointing from node v_i to v_j, where neighborhood of node v_i is defined as $N(v_i) = \{v_j; e_{ji} \in E\}$. Typically, GNNs use graph structure & node features to learn a node representation h_{v_i} through principle of neighborhood aggregation, where they iteratively update each node's representation by aggregating information of its neighbors. After undergoing k iterations of aggregation, node representation encapsulates structural information presented in its k-hop network vicinity: (1)

$$h_{v_i}^{(k)} = \mathcal{C}^{(k)}\left(h_{v_i}^{(k-1)}, \mathcal{A}^{(k)}\left(\left\{h_{v_j}^{(k-1)},\ \forall v_j \in N(v_i)\right\}\right)\right),$$

where $\mathcal{A}^{(k)}, \mathcal{C}^{(k)}$ are aggregation & combination operators at kth layer. They have distinct implementations for different neural networks.

– Do tính hiệu quả, tính tổng quát, & hiệu quả của GNN tích chập, mức độ phổ biến của nó đã tăng nhanh chóng trong 1 loạt các ứng dụng, bao gồm giải quyết các vấn đề tối ưu hóa tổ hợp NP-hard. Trong khảo sát này, chúng tôi thấy rằng các GNN tích chập thường được sử dụng để giải quyết các JSSP. Về mặt hình thức, 1 đồ thị được biểu diễn là $G = \{V, E\}$, trong đó V, E là các tập hợp các nút & cạnh, tương ứng. Giả sử $v_i \in V$ biểu thị 1 nnode, & $e_{ij} \in E$ biểu thị 1 cạnh trỏ từ nút v_i đến v_j, trong đó lân cận của nút v_i được định nghĩa là $N(v_i) = \{v_j; e_{ji} \in E\}$. Thông thường, các GNN sử dụng các đặc trưng cấu trúc đồ thị & nút để học biểu diễn nút h_{v_i} thông qua nguyên tắc tổng hợp lân cận, trong đó chúng cập nhật lặp lại biểu diễn của từng nút bằng cách tổng hợp thông tin của các nút lân cận của nó. Sau khi trải qua k lần lặp tổng hợp, biểu diễn nút sẽ đóng gói thông tin cấu trúc được trình bày trong vùng lân cận mạng k-hop của nó: (1)

$$h_{v_i}^{(k)} = \mathcal{C}^{(k)}\left(h_{v_i}^{(k-1)}, \mathcal{A}^{(k)}\left(\left\{h_{v_j}^{(k-1)},\ \forall v_j \in N(v_i)\right\}\right)\right),$$

trong đó $\mathcal{A}^{(k)}, \mathcal{C}^{(k)}$ là các toán tử tổng hợp & kết hợp ở lớp k. Chúng có các triển khai riêng biệt cho các mạng nơ-ron khác nhau.

- **Graph convolutional network (GCN).** (Kipf & Welling, 2017) is a powerful architecture motivated by a localized 1st-order approximation of spectral graph convolutions. Node representation matrix H is updated as follows: (2)

$$H^{(k)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k-1)}W^{(k)}),$$

where σ is an activation function, $\tilde{A} = A + I$ is adjacency matrix of undirected graph with added self-connections, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, & $W^{(k)}$ is a trainable parameter matrix at kth layer. Unfortunately, it does not naturally support edge features & directed graphs (e.g., disjunctive graphs).

– **Mạng tích chập đồ thị (GCN).** (Kipf & Welling, 2017) là 1 kiến trúc mạnh mẽ được thúc đẩy bởi phép xấp xỉ bậc 1 cục bộ của tích chập đồ thị phổ. Ma trận biểu diễn nút H được cập nhật như sau: (2)

$$H^{(k)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k-1)}W^{(k)}),$$

trong đó σ là 1 hàm kích hoạt, $\tilde{A} = A + I$ là ma trận kề của đồ thị vô hướng có thêm các kết nối tự thân, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, & $W^{(k)}$ là 1 ma trận tham số có thể huấn luyện được ở lớp thứ k. Thật không may, nó không hỗ trợ các tính năng cạnh & đồ thị có hướng (ví dụ: đồ thị rời rạc).

- **Graph attention network (GAT).** (Veličković et al., 2018) employs multi-head attention mechanisms from Transformer (Vaswani et al., 2017), with the update formulation as follows: (3)

$$h_{v_i}^{(k)} = ||_{m=1}^{M}\sigma\left(\sum_{v_j \in N(v_i)} \alpha_{ij}^m W_m^{(k)} h_{v_j}^{(k-1)}\right),$$

where $||$ is concatenation operator, M: number of attention heads, & α_{ij}: attention weight calculated by (4)

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}(W_2^\top [W_1 h_{v_i} || W_1 h_{v_j}])\right)}{\sum_{v_u \in N(v_i)} \exp\left(\text{LeakyReLU}(W_2^\top [W_1 h_{v_i} || W_1 h_{v_u}])\right)}.$$

By leaving out computing α_{ij} if e_{ji} is not present, GAT can deal with directed graphs.

– **Mạng lưới chú ý đồ thị (GAT).** (Veličković & cộng sự, 2018) sử dụng cơ chế chú ý đa đầu từ Transformer (Vaswani & cộng sự, 2017), với công thức cập nhật như sau: (3)

$$h_{v_i}^{(k)} = ||_{m=1}^{M} \sigma \left(\sum_{v_j \in N(v_i)} \alpha_{ij}^{m} W_m^{(k)} h_{v_j}^{(k-1)} \right),$$

trong đó $||$ là toán tử nối, M: số đầu chú ý, & α_{ij}: trọng số chú ý được tính theo (4)

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}(W_2^{\top}[W_1 h_{v_i} || W_1 h_{v_j}])\right)}{\sum_{v_u \in N(v_i)} \exp\left(\text{LeakyReLU}(W_2^{\top}[W_1 h_{v_i} || W_1 h_{v_u}])\right)}.$$

Bằng cách bỏ qua việc tính toán α_{ij} nếu không có e_{ji}, GAT có thể xử lý các đồ thị có hướng.

- **Graph isomorphism network (GIN).** (Xu et al., 2019) is another popular convolutional GNN variant with strong discriminative power for non-isomorphic graphs. Through theoretical analyses of expressive power of GNs, demonstrable: GIN stands out as most expressive among class of GNNs, matching power of Weisfeiler–Lehman graph isomorphism test. Concretely, it updates node representation as: (5)

$$h_{v_i}^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)}) h_{v_i}^{(k-1)} + \sum_{v_j \in N(v_i)} h_{v_j}^{(k-1)} \right),$$

where MLP is a Multi-Layer Perceptron, ϵ is an arbitrary number that can be learned. GIN is empirically found to be effective in solving JSSPs (Zhang et al., 2020, 2024). Most works leveraging GNNs to solve JSSPs directly apply above-mentioned architecture or design a variant tailored for each specific problem (but following same principle of neighborhood aggregation).

– **Mạng đồng cấu đồ thị (GIN).** (Xu & cộng sự, 2019) là 1 biến thể GNN tích chập phổ biến khác với khả năng phân biệt mạnh mẽ đối với đồ thị không đồng cấu. Thông qua các phân tích lý thuyết về khả năng biểu đạt của GN, có thể chứng minh: GIN nổi bật là mạng biểu đạt nhất trong số các lớp GNN, phù hợp với khả năng của phép thử đồng cấu đồ thị Weisfeiler–Lehman. Cụ thể, nó cập nhật biểu diễn nút như sau: (5)

$$h_{v_i}^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)}) h_{v_i}^{(k-1)} + \sum_{v_j \in N(v_i)} h_{v_j}^{(k-1)} \right),$$

trong đó MLP là 1 Perceptron đa lớp, ϵ là 1 số tùy ý có thể học được. GIN được chứng minh là hiệu quả trong việc giải quyết JSSP (Zhang & cộng sự, 2020, 2024). Hầu hết các công trình sử dụng GNN để giải quyết JSSP đều áp dụng trực tiếp kiến trúc đã đề cập ở trên hoặc thiết kế 1 biến thể phù hợp với từng bài toán cụ thể (nhưng tuân theo cùng nguyên tắc tổng hợp lân cận).

- 4. GNNs for JSSPs. There is a recent trend towards applying GNNs to solve JSSPs. Found: most works apply GNNs with DRL to avoid using optimal solutions as labels. Only a few works employ GNNs without DRL. In this sect, 1st summarize GNN-based methods without DRL & then introduce work on combining DRL & GNN to tackle a variety of JSSPs. At least, also review current GNN-based methods for solving flow-shop scheduling problems since they are natural extensions of methods for JSSPs.

  – Gần đây có 1 xu hướng ứng dụng GNN để giải quyết JSSP. Kết quả cho thấy: hầu hết các công trình đều áp dụng GNN với DRL để tránh sử dụng các giải pháp tối ưu làm nhãn. Chỉ 1 số ít công trình sử dụng GNN không có DRL. Trong phần này, trước tiên, chúng tôi tóm tắt các phương pháp dựa trên GNN không có DRL & sau đó giới thiệu các nghiên cứu về việc kết hợp DRL & GNN để giải quyết nhiều loại JSSP. Ít nhất, chúng tôi cũng xem xét các phương pháp dựa trên GNN hiện tại để giải quyết các bài toán lập lịch flow-shop vì chúng là phần mở rộng tự nhiên của các phương pháp dành cho JSSP.

  - 4.1. Non-DRL methods. Several studies have applied GNNs to solve JSSPs without relying on reinforcement learning (RL). Juros et al. (2022) evaluate a convolutional GNN-based method, originally proposed by Gasse et al. (2019), on JSSP & delivery scheduling problem (DSP). Proposed method uses a convolutional GNN to speed up variable selection procedure of branch-&-bound algorithm (B&B) in SCIP solver. Their approach employs imitation learning to approximate full strong branching (FSB) strategy of SCIP for a specific class of problems. This is achieved by solving multiple instances of same problem type in SCIP & recording state-action pairs, where actions are variables chosen for branching by FSB policy. These pairs are then used to train & test model, which leverages bipartite representations to model variables & constraints. Results indicate: model has partially succeeded in learning to imitate FSB policy. Default branching strategy of SCIP tends to be superior on JSSP, while learned model solves problem faster for 13 out of 34 instances on DSP. Authors also report high variance in model performance attributed to sampling distribution of problems.

    – 1 số nghiên cứu đã áp dụng GNN để giải JSSP mà không cần dựa vào học tăng cường (RL). Juros & cộng sự (2022) đánh giá 1 phương pháp dựa trên GNN tích chập, ban đầu được Gasse & cộng sự (2019) đề xuất, về JSSP & vấn đề lập lịch phân

phối (DSP). Phương pháp được đề xuất sử dụng GNN tích chập để tăng tốc quy trình lựa chọn biến của thuật toán ràng buộc nhánh (B&B) trong trình giải SCIP. Cách tiếp cận của họ sử dụng học bắt chước để xấp xỉ chiến lược phân nhánh mạnh (FSB) đầy đủ của SCIP cho 1 lớp vấn đề cụ thể. Điều này đạt được bằng cách giải quyết nhiều trường hợp cùng loại vấn đề trong SCIP & ghi lại các cặp trạng thái-hành động, trong đó các hành động là các biến được chọn để phân nhánh theo chính sách FSB. Các cặp này sau đó được sử dụng để huấn luyện & kiểm tra mô hình, tận dụng các biểu diễn hai phần để mô hình hóa các biến & ràng buộc. Kết quả cho thấy: mô hình đã thành công 1 phần trong việc học cách bắt chước chính sách FSB. Chiến lược phân nhánh mặc định của SCIP có xu hướng vượt trội hơn trên JSSP, trong khi mô hình học giải quyết vấn đề nhanh hơn với 13/34 trường hợp trên DSP. Các tác giả cũng báo cáo sự sai lệch lớn về hiệu suất mô hình do phân phối mẫu của các bài toán.

Similar to previous study, Lee & Kim (2022) employ imitation learning (i.e., behavior cloning) with a dynamic disjunctive graph representation for real-time JSSP. Method 1st generates dataset by utilizing optimal policies derived from optimal schedules obtained through constraint programming (CP). Dataset captures decision-making points (states) where idle machines have multiple potential (available & soon-to-be-ready) operations (actions) to perform. In this context, dynamic disjunctive graph adjusts at each decision point, reflecting only current & imminent operations. Building on this, 2 GNN models, adapted from Park et al. (2020, 2021b), are developed. 1st GNN utilizes an attention mechanism to aggregate node embeddings from neighbors connected via conjunctive & disjunctive edges. 2nd GNN, called multiplex, categorizes node features based on 3 connection types, including input conjunctive, output conjunctive, & disjunctive connections. It processes each category through separate layers & averages these embeddings after several aggregation iterations. Proposed approach is trained on custom-generated instances & tested on TA (Taillard, 1993) benchmark instances for which optimal solutions are known. Results show: imitation learning approach with disjunctive graphs & multiplex achieves lowest optimality gap on larger unseen (TA) instances.

– Tương tự như nghiên cứu trước đây, Lee & Kim (2022) sử dụng học bắt chước (tức là sao chép hành vi) với biểu diễn đồ thị phân ly động cho JSSP thời gian thực. Phương pháp thứ nhất tạo tập dữ liệu bằng cách sử dụng các chính sách tối ưu bắt nguồn từ các lịch trình tối ưu thu được thông qua lập trình ràng buộc (CP). Tập dữ liệu nắm bắt các điểm ra quyết định (trạng thái) trong đó các máy nhàn rỗi có nhiều hoạt động (hành động) tiềm năng (có sẵn & sắp sẵn sàng) để thực hiện. Trong bối cảnh này, đồ thị phân ly động điều chỉnh tại mỗi điểm quyết định, chỉ phản ánh các hoạt động & sắp xảy ra hiện tại. Dựa trên điều này, 2 mô hình GNN, được chuyển thể từ Park & cộng sự (2020, 2021b), đã được phát triển. GNN thứ nhất sử dụng cơ chế chú ý để tổng hợp các nhúng nút từ các nút lân cận được kết nối thông qua các cạnh & phân ly liên hợp. GNN thứ hai, được gọi là ghép kênh, phân loại các tính năng nút dựa trên 3 loại kết nối, bao gồm kết nối đầu vào liên hợp, kết nối đầu ra liên hợp, kết nối & phân ly. Nó xử lý từng danh mục thông qua các lớp riêng biệt & tính trung bình các nhúng này sau nhiều lần lặp tổng hợp. Phương pháp đề xuất được huấn luyện trên các trường hợp được tạo tùy chỉnh & được thử nghiệm trên các trường hợp chuẩn TA (Taillard, 1993) mà các giải pháp tối ưu đã biết. Kết quả cho thấy: phương pháp học mô phỏng với đồ thị rời rạc & ghép kênh đạt được khoảng cách tối ưu thấp nhất trên các trường hợp lớn hơn chưa được biết đến (TA).

Wang et al. (2023b) explore use of GNNs & graph transformer models to address combinatorial optimization problems (COPs), specifically focusing on JSSP & TSP. Their method employs a 2-step learning pipeline: initially, GNN (or a similar variant) processes a COP instance to generate embeddings, which are then utilized by a functional module to predict cost or makespan of optimal solution. For evaluation purposes, benchmark datasets are created for both problems, with JSSP dataset comprising problem sizes of $9 \times 9, 10 \times 10$, containing 10000 & 100000 instances resp., & TSP dataset including problems with 30 & 40 cities, each having 10000 instances. Findings indicate: GNN-based models significantly outperform previous models that used CNNs. However, paper does not include comparisons with other methods or heuristics. Additionally, authors suggest leveraging output from learning model to guide COP solvers in prioritizing search spaces close to predicted values, although no experimental results are presented to assess effectiveness of this strategy.

– Wang & cộng sự (2023b) khám phá việc sử dụng GNNs & mô hình biến đổi đồ thị để giải quyết các vấn đề tối ưu hóa tổ hợp (COP), đặc biệt tập trung vào JSSP & TSP. Phương pháp của họ sử dụng quy trình học 2 bước: ban đầu, GNN (hoặc 1 biến thể tương tự) xử lý 1 thể hiện COP để tạo ra các nhúng, sau đó được 1 mô-đun chức năng sử dụng để dự đoán chi phí hoặc khoảng thời gian của giải pháp tối ưu. Để đánh giá, các tập dữ liệu chuẩn được tạo cho cả hai vấn đề, với tập dữ liệu JSSP bao gồm các kích thước vấn đề là $9 \times 9, 10 \times 10$, tương ứng chứa 10000 & 100000 thể hiện, & tập dữ liệu TSP bao gồm các vấn đề với 30 & 40 thành phố, mỗi thành phố có 10000 thể hiện. Các phát hiện chỉ ra: các mô hình dựa trên GNN vượt trội đáng kể so với các mô hình trước đây sử dụng CNN. Tuy nhiên, bài báo không bao gồm các so sánh với các phương pháp hoặc phương pháp tìm kiếm khác. Ngoài ra, các tác giả đề xuất tận dụng kết quả từ mô hình học để hướng dẫn các trình giải COP ưu tiên các không gian tìm kiếm gần với các giá trị dự đoán, mặc dù không có kết quả thử nghiệm nào được trình bày để đánh giá hiệu quả của chiến lược này.

Corsini et al. (2024) propose a self-supervised learning method, called *self-labeling pointer network* (SPN) as an alternative to DRL, to create a solution construction agent for JSSPs. In this self-supervised training strategy, multiple parallel solutions are sampled for each problem instance. Then, best solution is used to provide pseudo-labels, & cross-entropy loss using these pseudo-labels is minimized. A pointer network using a GAT encoder & a multihead attention decoder is trained through this method. In specific, model is trained using instances generated through Taillard's method (Taillard, 1993) with sizes oof $10 \times 10, 15 \times 10, 15 \times 15, 20 \times 10, 20 \times 15, 20 \times 20$. Method is evaluated on TA & DMU (Demirkol et al., 1998) benchmark datasets. Their method outperforms priority dispatching rules (PDRs), insertion algorithm (Nowicki & Smutnicki, 1996), learning to dispatch (L2D) (Zhang et al., 2020), curriculum learning method (Iklassov et al., 2022), & enhanced local search metaheuristic (Falkner et al., 2022) by a clear margin. Especially when using sampling inference method, optimality gaps compared to L2D are more than half lower. Performance seems to match, or in some instances outperform, best DRL-based

methods for JSSP. Thus, this method may offer a promising direction for further research. However, due to autoregressive inference method without a Markov decision process (MDP) formulation, it may be somewhat limited to static problems. For dynamic problems, a purely autoregressive method may not be suitable without more advanced adaptations. Table 1: GNN-based non-DRL methods for JSSPs & its variants. Solution construction emphasizes policy network is not trained to act similarly to a PDR, as it is, e.g., trained for solving a single instance or in an autoregressive way. provides a summary of GNN-based methods for solving JSSP without using (D)RL.

– Corsini & cộng sự (2024) đề xuất 1 phương pháp học tự giám sát, được gọi là *self-labeling pointer network* (SPN) như 1 giải pháp thay thế cho DRL, để tạo ra 1 tác nhân xây dựng giải pháp cho JSSP. Trong chiến lược đào tạo tự giám sát này, nhiều giải pháp song song được lấy mẫu cho mỗi trường hợp vấn đề. Sau đó, giải pháp tốt nhất được sử dụng để cung cấp nhãn giả, & mất entropy chéo khi sử dụng các nhãn giả này được giảm thiểu. 1 mạng con trỏ sử dụng bộ mã hóa GAT & bộ giải mã chú ý nhiều đầu được đào tạo thông qua phương pháp này. Cụ thể, mô hình được đào tạo bằng các trường hợp được tạo thông qua phương pháp Taillard (Taillard, 1993) với kích thước oof $10 \times 10, 15 \times 10, 15 \times 15, 20 \times 10, 20 \times 15, 20 \times 20$. Phương pháp được đánh giá trên các tập dữ liệu chuẩn TA & DMU (Demirkol & cộng sự, 1998). Phương pháp của họ vượt trội hơn các quy tắc phân phối ưu tiên (PDR), thuật toán chèn (Nowicki & Smutnicki, 1996), học cách phân phối (L2D) (Zhang & cộng sự, 2020), phương pháp học theo chương trình giảng dạy (Iklassov & cộng sự, 2022), & siêu thuật toán tìm kiếm cục bộ nâng cao (Falkner & cộng sự, 2022) với biên độ rõ ràng. Đặc biệt khi sử dụng phương pháp suy luận lấy mẫu, khoảng cách tối ưu so với L2D thấp hơn 1 nửa. Hiệu suất dường như phù hợp hoặc trong 1 số trường hợp vượt trội hơn các phương pháp dựa trên DRL tốt nhất cho JSSP. Do đó, phương pháp này có thể cung cấp 1 hướng đầy hứa hẹn cho nghiên cứu sâu hơn. Tuy nhiên, do phương pháp suy luận tự hồi quy không có công thức quy trình quyết định Markov (MDP), nên nó có thể bị giới hạn phần nào đối với các vấn đề tĩnh. Đối với các vấn đề động, phương pháp tự hồi quy thuần túy có thể không phù hợp nếu không có các điều chỉnh nâng cao hơn. Bảng 1: Các phương pháp không phải DRL dựa trên GNN cho JSSP & các biến thể của nó. Giải pháp xây dựng nhấn mạnh rằng mạng chính sách không được đào tạo để hoạt động tương tự như PDR, vì nó được đào tạo để giải quyết 1 trường hợp duy nhất hoặc theo cách tự hồi quy. cung cấp tóm tắt về các phương pháp dựa trên GNN để giải quyết JSSP mà không sử dụng (D)RL.

- 4.2. DRL methods. By formulating JSSPs as sequential decision-making problems, most existing research on JSSPs leverages GNNs & RL algorithms. On 1 hand, GNN is used to learn a policy to iteratively construct solutions, which is capable of capturing favorable representations of JSSP instances & partial solutions from topology of graphs. On other hand, RL algorithms feature interactions between agent (parameterized by GNN) & environment, with trajectories (i.e., sequences of interactions) collected to update agent. In comparison to supervised learning, RL is beneficial in avoiding usage of labels, which are optimal solutions & hard to achieve usage of labels, which are optimal solutions & hard to achieve due to high computational complexity. In following secs, 1st introduce preliminaries on DRL, & then elaborate on existing work on applying GNNs & DRL to typical JSSPs. In this paper, mainly focus on basic JSSP, flexible JSSP, dynamic JSSP, distributed JSSP, flow-shop scheduling problem (FSP), hybrid flow-shop scheduling problem (HFSP), & permutation flow-shop scheduling problem (PFSP). Since there is little research on other types of JSSPs, do not cover those problems in this survey.

  – Bằng cách xây dựng JSSP như các bài toán ra quyết định tuần tự, hầu hết các nghiên cứu hiện có về JSSP đều tận dụng các thuật toán GNN & RL. 1 mặt, GNN được sử dụng để học 1 chính sách nhằm xây dựng các giải pháp theo cách lặp lại, có khả năng nắm bắt các biểu diễn thuận lợi của các thể hiện JSSP & các giải pháp 1 phần từ tôpô của đồ thị. Mặt khác, các thuật toán RL có các tương tác giữa tác nhân (được tham số hóa bởi GNN) & môi trường, với các quỹ đạo (tức là các chuỗi tương tác) được thu thập để cập nhật tác nhân. So với học có giám sát, RL có lợi trong việc tránh sử dụng nhãn, vốn là các giải pháp tối ưu & khó đạt được do độ phức tạp tính toán cao. Trong các phần tiếp theo, trước tiên hãy giới thiệu sơ bộ về DRL, & sau đó trình bày chi tiết về các công trình hiện có về việc áp dụng GNN & DRL vào các JSSP thông thường. Bài viết này tập trung chủ yếu vào JSSP cơ bản, JSSP linh hoạt, JSSP động, JSSP phân tán, bài toán lập lịch flow-shop (FSP), bài toán lập lịch flow-shop lai (HFSP), bài toán lập lịch flow-shop hoán vị (PFSP). Do có ít nghiên cứu về các loại JSSP khác, nên bài khảo sát này sẽ không đề cập đến các vấn đề đó.

  * 4.2.1. Preliminaries on DRL. DRL combines traditional RL with DL techniques to effectively tackle sequential decision-making problems. Given an MDP formulation of a sequential decision-making problem, an agent sequentially takes actions according to varying states derived from environment. Environment iteratively responds to agent's actions by providing rewards & transitioning to new states. Objective of DRL: find an optimal policy π_θ^* that maximized expected cumulative reward (i.e., return). Generally, policy-based & value-based approaches are 2 primary methods for learning optimal policies. Policy-based approaches, e.g. REINFORCE, A3C, & PRO (Schulman et al., 2017; Williams, 1992; Mnih et al., 2016), directly learns policy π_θ, which defines probability distribution over actions for each state. In contrast, value-based approaches, e.g. DQN, double DQN, & dueling DQN (Wang et al., 2016; Mnih et al., 2015; Van Hasselt et al., 2016), focuses on learning value functions Q_δ^*, which estimate expected cumulative reward for each state (or state–action pair). Refer interested readers to François-Lavet et al. (2018) & Tassel et al. (2021) for more knowledge on DRL.

    – Sơ bộ về DRL. DRL kết hợp RL truyền thống với các kỹ thuật DL để giải quyết hiệu quả các vấn đề ra quyết định tuần tự. Với 1 công thức MDP của 1 vấn đề ra quyết định tuần tự, 1 tác nhân sẽ thực hiện các hành động tuần tự theo các trạng thái khác nhau bắt nguồn từ môi trường. Môi trường phản ứng lặp lại với các hành động của tác nhân bằng cách cung cấp phần thưởng & chuyển sang các trạng thái mới. Mục tiêu của DRL: tìm 1 chính sách tối ưu π_θ^* để tối đa hóa phần thưởng tích lũy kỳ vọng (tức là lợi nhuận). Nhìn chung, các phương pháp tiếp cận dựa trên chính sách & dựa trên giá trị là 2 phương pháp chính để học các chính sách tối ưu. Các phương pháp tiếp cận dựa trên chính sách, ví dụ: REINFORCE, A3C, & PRO (Schulman & cộng sự, 2017; Williams, 1992; Mnih & cộng sự, 2016), học trực tiếp chính sách π_θ, xác định phân phối xác suất trên các hành động cho mỗi trạng thái. Ngược lại, các phương pháp tiếp cận dựa trên

giá trị, ví dụ: DQN, DQN kép, DQN đấu tay đôi (Wang & cộng sự, 2016; Mnih & cộng sự, 2015; Van Hasselt & cộng sự, 2016), tập trung vào việc học các hàm giá trị Q^*_δ, ước tính phần thưởng tích lũy kỳ vọng cho mỗi trạng thái (hoặc cặp trạng thái-hành động). Mời độc giả quan tâm tham khảo François-Lavet & cộng sự (2018) & Tassel & cộng sự (2021) để biết thêm thông tin về DRL.

∗ 4.2.2. MDP formulations of JSSPs. MDP formulations for JSSPs are varying in different methods. Here, introduce 1 commonly used MDP formulation for learning construction heuristics in most existing literature (Zhang et al., 2020). In MDP:

1. *state* is problem instance & partial schedule updated in solving process, which is depicted by graph representation introduced in Sect. 2;
2. given current state, *action* set is dispatching operations to their designated machines;
3. *transition* is a deterministic function, which changes state (i.e., graph representation) according to dispatching action (e.g., a new link can be added in graph to represent an operation is dispatched after another operation on same machine);
4. *reward* can be defined as difference between makespan of current state & previous state, with return of a trajectory equivalent to total makespan (i.e., objective);
5. stochastic *policy* takes as input graph representation of state, & outputs feasible operations to be dispatched to a machine. On top of above general MDP, various MDPs are also proposed for different types of JSSPs with slight adjustments. E.g., commonly used MDP of FJSSPs only differs slightly from JSSPs in state & action set. Specifically, state representation includes additional machine nodes, often depicted as extra nodes in a heterogeneous graphs (Song et al., 2022). Actions, in turn, consist of a selected operation-machine pair, rather than merely an operation, after which corresponding links are adjusted in transition function.

– Công thức MDP của JSSP. Công thức MDP cho JSSP có nhiều phương pháp khác nhau. Dưới đây, xin giới thiệu 1 công thức MDP thường được sử dụng để học các thuật toán xây dựng trong hầu hết các tài liệu hiện có (Zhang & cộng sự, 2020). Trong MDP:

1. *state* là thể hiện bài toán & lịch trình 1 phần được cập nhật trong quá trình giải, được mô tả bằng biểu diễn đồ thị được giới thiệu trong Phần 2;
2. với trạng thái hiện tại, *action* là tập hợp các thao tác được phân phối đến các máy được chỉ định;
3. *transition* là 1 hàm xác định, thay đổi trạng thái (tức là biểu diễn đồ thị) theo hành động phân phối (ví dụ: có thể thêm 1 liên kết mới vào đồ thị để biểu diễn 1 thao tác được phân phối sau 1 thao tác khác trên cùng 1 máy);
4. *reward* có thể được định nghĩa là sự khác biệt giữa khoảng thời gian hoàn thành của trạng thái hiện tại & trạng thái trước đó, với kết quả trả về là 1 quỹ đạo tương đương với khoảng thời gian hoàn thành tổng thể (tức là mục tiêu);
5. stochastic *policy* lấy biểu diễn đồ thị trạng thái làm đầu vào, & đưa ra các thao tác khả thi để phân phối đến máy. Ngoài MDP chung nêu trên, nhiều MDP khác nhau cũng được đề xuất cho các loại JSSP khác nhau với 1 số điều chỉnh nhỏ. Ví dụ, MDP thường được sử dụng của FJSSP chỉ khác 1 chút so với JSSP ở tập trạng thái & hành động. Cụ thể, biểu diễn trạng thái bao gồm các nút máy bổ sung, thường được mô tả là các nút bổ sung trong đồ thị không đồng nhất (Song & cộng sự, 2022). Các hành động, đến lượt nó, bao gồm 1 cặp thao tác-máy được chọn, thay vì chỉ là 1 thao tác, sau đó các liên kết tương ứng được điều chỉnh trong hàm chuyển tiếp.

In DRL-based methods, GNNs play important roles in learning policies for constructing solutions. Various GNNs have been proposed for solving JSSPs with DRL. In following sects, elaborate on how these GNNs are uesd in each study concerning JSSPs.

– Trong các phương pháp dựa trên DRL, GNN đóng vai trò quan trọng trong việc học các chính sách để xây dựng giải pháp. Nhiều GNN khác nhau đã được đề xuất để giải quyết JSSP bằng DRL. Trong các phần sau, chúng tôi sẽ trình bày chi tiết cách sử dụng các GNN này trong từng nghiên cứu liên quan đến JSSP.

∗ 4.2.3. Basic JSSPs. Zhang et al. (2020) are 1st to propose a DR approach, called L2D, to solve JSSPs. They propose to learn priority dispatching rules based on disjunctive graph representation. They use a GIN network trained with PPO to learn embeddings for disjunctive graphs & select operations to schedule. Proposed method is trained & evaluated on instances generated by Taillard's method with sizes ranging between $6\times6, 30\times20$. In addition, they report their evaluation results on TA & DMU benchmark instances up to 100×20. Their method outperforms priority dispatching rules while maintaining a fast speed. Method underperforms compared to exact solutions obtained through OR-Tools CP-SAT solver (Perron & Didier, 2024), but it operates much faster for larger problems. Park et al. (2021b) propose GNNRL, a similar method to L2D. Instead of a GIN network, they design a custom graph embedding layer that incorporates a basic message passing structure, which accounts for different edge types. For training, they generate instances following uniform distribution of 5 to 9 machines & up to 9 jobs with uniform processing times between 1 to 99 time units. They evaluate with similar instances, as well as ORB (Applegate & Cook, 1991), SWV (Storer et al., 1992), LA (Lawrence, 1984), ABZ (Adams et al., 1988), & YN (Yamada & Nakano, 1992) benchmarks, which have sizes from 5 to 20 machines & 6–100 jobs. Their method shows better performance than all PDRs & good generalization to different benchmarks with varying sizes.

– JSSP cơ bản. Zhang & cộng sự (2020) là những người đầu tiên đề xuất phương pháp DR, được gọi là L2D, để giải quyết JSSP. Họ đề xuất học các quy tắc phân phối ưu tiên dựa trên biểu diễn đồ thị rời rạc. Họ sử dụng mạng GIN được đào tạo với PPO để học các nhúng cho đồ thị rời rạc & chọn các hoạt động để lên lịch. Phương pháp đề xuất được đào tạo & đánh giá trên các trường hợp được tạo bởi phương pháp Taillard với kích thước nằm trong khoảng $6\times6, 30\times20$. Ngoài ra, họ báo cáo kết quả đánh giá của mình trên các trường hợp chuẩn TA & DMU lên đến 100×20. Phương pháp của họ vượt trội hơn các quy tắc phân phối ưu tiên trong khi vẫn duy trì tốc độ nhanh. Phương pháp này kém hiệu quả hơn so

với các giải pháp chính xác thu được thông qua bộ giải CP-SAT của OR-Tools (Perron & Didier, 2024), nhưng nó hoạt động nhanh hơn nhiều đối với các vấn đề lớn hơn. Park & cộng sự (2021b) đề xuất GNNRL, 1 phương pháp tương tự như L2D. Thay vì mạng GIN, họ thiết kế 1 lớp nhúng đồ thị tùy chỉnh kết hợp cấu trúc truyền tin cơ bản, tính đến các loại cạnh khác nhau. Để huấn luyện, họ tạo ra các trường hợp sau khi phân phối đồng đều từ 5 đến 9 máy & tối đa 9 tác vụ với thời gian xử lý đồng đều từ 1 đến 99 đơn vị thời gian. Họ đánh giá bằng các trường hợp tương tự, cũng như các chuẩn ORB (Applegate & Cook, 1991), SWV (Storer & cộng sự, 1992), LA (Lawrence, 1984), ABZ (Adams & cộng sự, 1988) & YN (Yamada & Nakano, 1992), có quy mô từ 5 đến 20 máy & 6-100 tác vụ. Phương pháp của họ cho thấy hiệu suất tốt hơn tất cả các PDR & khả năng khái quát hóa tốt cho các chuẩn khác nhau với quy mô khác nhau.

Hottung et al. (2022) propose an efficient active search (EAS) to improve DL policies for combinatorial optimization. This method adjusts a subset of model parameters on an individual instance in order to search for better solutions. They test their method on several combinatorial optimization problems, including JSSPs. In specific, they use EAS to enhance performance of L2D policies. Their method considerably improves performance over naive sampling methods at cost of increased runtime. As EAS is generally applicable, it can be used to improve any of discussed DRL methods in this paper.

Chalumeau et al. (2023) also propose an alternative method to improve existing policies in combinatorial optimization based on latent space search. This method, called COMPASS, conditions networks on a latent space. Then, latent space is searched using evolutionary strategies to obtain better policies. Although additional training is needed for COMPASS, inference time barely increases, unlike EAS. In addition, COMPASS achieves better performance than EAS. Hence, it provides another alternative to improve existing policies.

– Hottung & cộng sự (2022) đề xuất 1 phương pháp tìm kiếm chủ động (EAS) hiệu quả để cải thiện các chính sách DL cho tối ưu hóa tổ hợp. Phương pháp này điều chỉnh 1 tập con các tham số mô hình trên 1 cá thể riêng lẻ để tìm kiếm các giải pháp tốt hơn. Họ đã thử nghiệm phương pháp của mình trên 1 số bài toán tối ưu hóa tổ hợp, bao gồm cả JSSP. Cụ thể, họ sử dụng EAS để nâng cao hiệu suất của các chính sách L2D. Phương pháp của họ cải thiện đáng kể hiệu suất so với các phương pháp lấy mẫu ngây thơ với chi phí tăng thời gian chạy. Vì EAS có thể áp dụng rộng rãi, nó có thể được sử dụng để cải thiện bất kỳ phương pháp DRL nào đã thảo luận trong bài báo này. Chalumeau & cộng sự (2023) cũng đề xuất 1 phương pháp thay thế để cải thiện các chính sách hiện có trong tối ưu hóa tổ hợp dựa trên tìm kiếm không gian tiềm ẩn. Phương pháp này, được gọi là COMPASS, đặt điều kiện mạng trên 1 không gian tiềm ẩn. Sau đó, không gian tiềm ẩn được tìm kiếm bằng các chiến lược tiến hóa để có được các chính sách tốt hơn. Mặc dù COMPASS cần được đào tạo thêm, thời gian suy luận hầu như không tăng, không giống như EAS. Ngoài ra, COMPASS đạt được hiệu suất tốt hơn EAS. Do đó, nó cung cấp 1 giải pháp thay thế khác để cải thiện các chính sách hiện có.

In contrast to previous methods, Park et al. (2021a) consider JSSP as a multi-agent problem, with machines representing agents. They propose ScheduleNet, which is a general scheduler for solving multi-agent scheduling problems. This approach utilizes an agent-task graph to model relationships & employs a type-aware graph attention network to extract information. This method facilitates efficient scaling to larger settings. They uniformly sample instances for training with 7–14 jobs & 2–5 machines. Method is evaluated on randomly generated instances ranging from size 6×6 to 100×20, as well as ORB, SWV, FT, LA, YN, & TA benchmarks. ScheduleNet outperforms previous methods (Zhang et al., 2020; Park et al., 2021b), while being a generalizable method. In addition, for 100×20 instances, they outperform CP-SAT with a 1-h time limit.

– Ngược lại với các phương pháp trước đây, Park & cộng sự (2021a) coi JSSP là 1 bài toán đa tác nhân, với các máy đại diện cho các tác nhân. Họ đề xuất ScheduleNet, 1 bộ lập lịch chung để giải quyết các bài toán lập lịch đa tác nhân. Phương pháp này sử dụng đồ thị tác nhân-nhiệm vụ để mô hình hóa các mối quan hệ & sử dụng mạng chú ý đồ thị nhận biết kiểu để trích xuất thông tin. Phương pháp này tạo điều kiện cho việc mở rộng hiệu quả sang các thiết lập lớn hơn. Họ lấy mẫu đồng đều các trường hợp để đào tạo với 7-14 công việc & 2-5 máy. Phương pháp được đánh giá trên các trường hợp được tạo ngẫu nhiên trong phạm vi kích thước từ 6×6 đến 100×20, cũng như các điểm chuẩn ORB, SWV, FT, LA, YN, & TA. ScheduleNet vượt trội hơn các phương pháp trước đây (Zhang & cộng sự, 2020; Park & cộng sự, 2021b), đồng thời là 1 phương pháp có thể khái quát hóa. Ngoài ra, đối với 100×20 trường hợp, chúng vượt trội hơn CP-SAT với giới hạn thời gian 1 giờ.

- 5. Discussion.
- 6. Potential research directions.
- 7. Conclusion.

5 Miscellaneous

Tài liệu

[KV18] Bernhard Korte and Jens Vygen. *Combinatorial optimization*. Vol. 21. Algorithms and Combinatorics. Theory and algorithms, Sixth edition of [MR1764207]. Springer, Berlin, 2018, pp. xxi+698. ISBN: 978-3-662-56038-9; 978-3-662-56039-6. DOI: 10.1007/978-3-662-56039-6. URL: https://doi.org/10.1007/978-3-662-56039-6.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.