Combinatorics – Tổ Hợp

Nguyễn Quản Bá Hồng*

Ngày 8 tháng 6 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:

- *Combinatorics – Tổ Hợp*.
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/NQBH_combinatorics.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/combinatorics/NQBH_combinatorics.tex.

Mục lục

1 Combinatorics – Tổ Hợp **1**
1.1 [Sha22]. Shahriar Shahriari. An Invitation to Combinatorics 1

2 Graph Theory – Lý Thuyết Đồ Thị **5**
2.1 [Val02; Val21]. Gabriel Valiente. Algorithms on Trees & Graphs With Python Code 5

3 Optimization on Graphs **7**
3.1 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning 7
3.2 [Gol18]. Boris Goldengorin. Optimization Problems in Graph Theory. 2018 15

4 Wikipedia's **15**
4.1 Wikipedia/extremal combinatorics 15
4.2 Wikipedia/extremal graph theory 15
4.2.1 History 15
4.2.2 Topics & concepts 16

5 Miscellaneous **16**

Tài liệu **16**

1 Combinatorics – Tổ Hợp

1.1 [Sha22]. Shahriar Shahriari. An Invitation to Combinatorics

[14 Amazon ratings]

- Preface. Combinatorics is a fun, difficult, broad, & very active area of mathematics. Counting, deciding whether certain configurations exist, & elementary graph theory are where subject begins. There are a myriad of connections to other areas of mathematics & CS, &, in fact, combinatorial problems can be found almost everywhere. To learn combinatorics is partly to become familiar with combinatorial topics, problems, & techniques, & partly to develop a can-do attitude toward discrete problem solving. This textbook is meant for a student who has completed an introductory college calculus sequence (a few sects require some knowledge of linear algebra but you can get quite a bit out of this text without a thorough understanding of linear algebra), has some familiarity with proofs, & desires to not only become acquainted with main topics of introductory combinatorics but also to become a better problem solver.
  - Key Features.

*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.

- **Conversational Style.** This text is written for students & is meant to be read. Reading mathematics is difficult, but being able to decipher complicated technical writing is an incredibly useful & transferable skill. Discussion in between usual theorems, proofs, & examples are meant to facilitate reader's (ad)venture into reading a mathematics textbook.
- **Problem-Solving Emphasis.** 1 advantage of combinatorics: many of its topics can be introduced without too much jargon. Can turn to almost any chap in this book & find problem statements that are understandable regardless of your background. Initially, may not know how to do a problem or even where to begin, but gaining experience in passing that hurdle is at heart of becoming a better problem solver. For this to happen, reader have to get actively involved, & learn by solving problems. Getting right answer is not really objective. Rather, it is only by trying to solve a problem that you will really understand what problem is asking & what subtle issues need to be considered. It is only after you have given problem a try that you will appreciate solution. This text gives you ample opportunity to get actively involved. In addition to > 1200 problems, highlight following features.
  - **Collaborative Mini-projects.** Mini-projects – there are 10 of these scattered throughout text – are meant to be projects for groups of 3 or 4 students to collaboratively explore. They are organized akin to a science lab. A few preliminary problems are to be done individually. Collaborative part of project is meant to take up good part of an afternoon, & it is envisioned that a project report – really a short mathematics paper – will be result. Mini-projects are of 2 types. In 1 type (Mini-projects 1, 6–10), project explores new material not covered elsewhere in text. 2nd type (Mini-projects 2–5) are meant to be done by students *before* relevant topic is covered in class. (This should explain their curious placement in text). It has been my experience: much learning happens if students 1st work on a topic, in a guided & purposeful manner, on their own, followed by class discussion/lecture.
  - **Guided Discovery through Scaffolded Projects.** In addition to collaborative mini-projects, quite a number of other topics are organized as a sequence of manageable smaller problems. Often assign these problems in consecutive assignments, so that can provide solutions, & allow time for discussion before proceeding to subsequent steps. In some other problems, students are guided toward a solution through an explicit sequence of steps. Some of proofs are presented in this format with hope: an engaged reader, with a paper & pencil at hand, will fill in details. As such, these problems are aimed at training reader in art of reading terse mathematical proofs.
  - **Warm-Up Problems & Opening Chap Problems.** Nearly every sect starts with a warm-up problem. These are relatively straightforward problems, & use them for in-class group work as a prelude to discussion of a topic. In contrast, every chap starts with a more subtle opening problem. Guess: for most part, student will not be able to do these opening problems before working through chap. Purpose: give a glimpse of what we are going to do in chap, & motivate to make it through material. Each opening chap problem is solved somewhere in chap, & all of warm-up problems have a short answer in Appendix A.
  - **Selected Hints, Short Answers, Complete Solutions.** Appendices have hints, short answers, & complete solutions for selected problems. Try a problem 1st without looking at solution, but when you are stuck – & hopefully you will get stuck often & learn to cherish experience – then 1st look at hint sect, then later at short answer sect, & finally at complete solutions sect. Complete solutions serve 2 purposes. They provide further examples of how to do problems, & they model how to write mathematics in paragraph style. By contrast, short answers are not particularly helpful in telling you how to do a problem. Instead, after done with a problem, short answer can either reassure you or send you back to drawing board.
  - **Open Problems & Conjectures.** A number of chaps end with a sect highlighting a few easily stated open problems & conjectures. Some of these are important unsolved problems, while others are mere curiosities. This is not meant to be a guide to current cutting-edge research problems. Rather, modest aim: whet readers' appetite by convincing her: even some seemingly innocent-looking problems remain unsolved, & combinatorics is an active area of research.
  - **Historical Asides.** I am not a historian & historical comments & footnotes barely scratch surface. Even so, combinatorial problems & solutions are a wonderful example of international nature of mathematics. In addition, mathematics is created by humans who are affected by, participate in, & sometimes, for good or bad, help shape communities & societies that they are a part of. Declare success even if just a few of you become curious & further investigate historical context of mathematics. Have also chosen to highlight international nature of combinatorics by naming some well-known mathematical objects differently.

- Coverage & Organization. Text has more than enough material for a 1-semester course in combinatorics at sophomore or junior level at an American university. Sects of book that I do not cover in my classes, & consider optional, are marked by a $\star$. Induction proofs & recurrence relations will be used throughout book & subject of Chap. 1. Counting problems – so-called *enumerative combinatorics* – take up $>$ half of book, & are subject of Chaps. 3–9. In Chap. 3, introduce a slew of "balls & boxes" problems that serve as an organizing principle for our counting problems. Chaps. 3–5 cover basics of permutations & combinations as well as a good dose of exploration of binomial coefficients. Chaps. 6–7 are on Stirling numbers & integer partitions. 2 substantial chaps on inclusion-exclusion principle & generating functions conclude our treatment of enumerative combinatorics. Graph theory is about $\frac{1}{3}$ of book & is covered in Chaps. 2 & 10. Basic vocabulary of graphs is introduced early in Sect. 2.2, but, for most part, material on graph theory is independent of other chaps. Start course with Ramsey theory since want to make sure: all students are seeing sth new, & they are not lulled into thinking: class is going to be only about permutations & combinations. But Ramsey theory is difficult & could be postponed to much later. Alternatively, could start with Chap. 10, & do graph theory 1st. Finally, Chap. 11 brings together material on partially ordered sets, a favorite of mine. Matchings in bipartite graphs is also covered in this chap, since wanted to bring out close connection between 2 frameworks.

Instructor may want to augment usual fare of introductory combinatorics with 1 or 2 more substantial results. Among topics offered here are Chung–Feller theorem, Euler's pentagonal number theorem, Cayley's theorem on labeled trees, Stanley's theorem on acyclic orientations, Thomassen's 5-color theorem, Pick's formula, Erdöds–Ko–Rado theorem, Ramsey's theorem for hypergraphs, & Möbius inversion.

- Global Roots of Combinatorics & Naming of Mathematical Objects. Most "new" mathematical ideas & concepts have antecedents & precursors in older ones, &, as a result, search for "1st" appearance of this or that mathematics is never-ending & often futile. As such naming of mathematical objects & results sometimes – possibly always – is a bit arbitrary

  > "It takes a thousand men to invent a telegraph or a steam engine, or a phonograph, or a telephone, or any other important thing – & last man gets credit & forget others. He added his little mite – that is all he did. These object lessons should teach us that 99 parts of all things that proceed from the intellect are plagiarisms, pure & simple; & lesson ought to make us modest. But nothing can do that." – Mark Twain, *Letter to Helen Keller, Riverdale-on-the-Hudson*, St. Patrick's Day 1903.
  >
  > – "Cần cả ngàn người để phát minh ra máy điện báo hay máy hơi nước, hay máy hát, hay điện thoại, hay bất kỳ thứ quan trọng nào khác – & người cuối cùng được ghi công & quên mất những người khác. Ông ấy đã thêm 1 đồng nhỏ của mình – đó là tất cả những gì ông ấy đã làm. Những bài học thực tế này sẽ dạy chúng ta rằng 99 phần của tất cả mọi thứ xuất phát từ trí tuệ đều là đạo văn, thuần túy & đơn giản; & bài học sẽ khiến chúng ta khiêm tốn. Nhưng không gì có thể làm được điều đó."

  However, when look at totality of common names of mathematical objects in combinatorics – Pascal's triangle, Vandermonde's identity, Catalan numbers, Stirling numbers, Bell numbers, or Fibonacci numbers – a remarkable & seemingly non-random pattern emerges. All names chosen are from European tradition. Undoubtedly, European mathematicians contributed significantly – &, in many subareas of mathematics, decisively – to development of mathematics. However, this constellation of names conveys to beginning student that combinatorial ideas & investigations were limited to Europe. In case of combinatorics, nothing could further from truth. Mathematicians from China, Japan, India, Iran, northern Africa, wider Islamic world, & Hebrew tradition, to mention a few, have very much worked on these topics. (For some of this history, see [Wilson & Watkins 2013]). Certainly, later European scholars have taken some topics further, but this does not take away from international character of mathematics in general & combinatorics in particular. For this reason, have tried to use a more inclusive set of names for at least some of familiar objects. Completely understandable to want to be familiar with more common – often universally accepted – names for various objects, & those are pointed out in text as well. Author does not claim expertise in history of combinatorics, & quite possible: every good historical arguments can be made in support of other attributions or against ones suggested here. If such a discussion ensues, will all be better for it.

  Combinatorics is a fertile area for involving undergraduates in research.

- Introduction.

  > "Accurate reckoning. The entrance into knowledge of all existing things & all obscure secrets." – The Ahmes–Rhind Papyrus

  - What is Combinatorics? Combinatorics is a collection of techniques & a language for study of (finite or countably infinite) discrete structures. Given a set of elements (& possibly some structure on that set), typical questions in combinatorics are:
    - Does a specific arrangement of elements exist?
    - How many such arrangements are there?
    - What properties do these arrangements have?
    - Which 1 of arrangements is maximal, minimal, or optimal according to some criterion?

    Unlike many other areas of mathematics – e.g., analysis, algebra, topology – core of combinatorics is neither its subject matter nor a set of "fundamental" theorems. More than anything else, combinatorics is a collection – some may say a hodgepodge – of techniques, attitudes, & general principles for solving problems about discrete structures. For any given problem, a combinatorist combines some of these techniques & principles – e.g., pigeonhole principle, inclusion-exclusion principle, marriage theorem, various counting techniques, induction, recurrence relations, generating functions, probabilistic arguments, asymptotic analysis – with (often clever) ad hoc arguments. Result is a fun & difficult subject.

    In today's mathematical world, in no small part due to power of digital computers, most mathematicians find much use for tool box of combinatorics. In problems of pure mathematics, often, after deciphering layers of theory, find a combinatorics problem at core. Outside of mathematics, e.g., combinatorial problems abound in CS.

  - Typical Problems. To whet your appetite, a preliminary sample of problems that we will encounter in course of this text.
    - How many sequences $a_1, \ldots, a_{12}$ are there consisting of 4 0's & 8 1's, if no 2 consecutive terms are both 0's?
    - A bakery has 8 kinds of donuts, & a box holds 1 dozen donuts. How many different boxes can you buy? How many different boxes are there that contain at least 1 of each kind?
    - A bakery sells 7 kinds of donuts. How many ways are there to choose 1 dozen donuts if no more than 3 donuts of any kind are used?
    - Determine number of n-digit numbers with all digits odd, s.t. 1 & 3 each occur a *positive* even number of times.

    - Try to reconstruct a word made from letters A, B, C, D, & R. Given a frequency table that shows number of times a specific triple occurs in word. [Table: triple: frequency]. Want to know all words with same triples & with same frequency table. Answer may be: there are no such words. Note: by a word we mean an ordered collection of letters & not concerned with meaning.
    - A soccer ball is usually tiled with 12 pentagons & 20 hexagons. Are any other combinations of pentagons & hexagons possible?
  - How Do We "Count"? Counting number of configurations of a certain type is an important part of combinatorics. In all of examples in prev sect, clear what kind of an answer we are looking for. Want a specific numerical answer or an example of a specific configuration.

    However, in many problems, it may be possible to present a solution satisfactory in many ways but is not quite a direct answer.

    – Tuy nhiên, trong nhiều bài toán, có thể đưa ra giải pháp thỏa đáng theo nhiều cách nhưng không phải là câu trả lời trực tiếp.

    Unclear how irrational numbers got involved in counting a discrete phenomenon. This formula can actually be used but seems to give little insight into problem. Sometimes, there are alternatives to finding a closed formula.

    As examples show, will not only use a myriad of techniques for solving counting problems, but will also refine our sense of what a good situation should look like. This all will (hopefully) become clear as we get our hands dirty & start solving problems.

    – Như các ví dụ cho thấy, sẽ không chỉ sử dụng vô số kỹ thuật để giải quyết các vấn đề đếm, mà còn tinh chỉnh cảm nhận của chúng ta về tình huống tốt nên như thế nào. Tất cả những điều này (hy vọng) sẽ trở nên rõ ràng khi chúng ta bắt tay vào giải quyết vấn đề.

- 1. Introduction & Recurrence Relations. Induction is a powerful method of proof & immensely useful in combinatorics. Suspect: most readers already have some experience with induction, & so 1st 2 sects of this chap should provide a quick review & some additional practice. Recurrence relations are ubiquitous in combinatorics & provide another powerful tool in analyzing counting problems. This will be important through text, but Sect. 1.3 gives experience constructing & using recurrence relations. Often recurrence relations & induction provide a 1-2 punch. You are interested in a sequence of integers – maybe a sequence that arises from a counting problem – so 1st find a recurrence relation for sequence, then you use it to generate data, & finally use induction to prove any patterns that you find. If you have prior experience with induction & recurrence relations, then you should try some of problems & move quickly to subsequent chaps. However, gaining experience with setting up recurrence relations by doing problems – maybe concurrently as you work on later chaps – is highly recommended. Optional Sect. 1.4 introduces to 2 possible methods for attacking recurrence relations.

  – Quy nạp là 1 phương pháp chứng minh mạnh mẽ & vô cùng hữu ích trong tổ hợp học. Nghi ngờ: hầu hết độc giả đã có 1 số kinh nghiệm với quy nạp, & vì vậy 2 phần đầu tiên của chương này sẽ cung cấp 1 bản tóm tắt nhanh & 1 số bài tập thực hành bổ sung. Các quan hệ đệ quy có mặt ở khắp mọi nơi trong tổ hợp & cung cấp 1 công cụ mạnh mẽ khác để phân tích các bài toán đếm. Điều này sẽ quan trọng thông qua văn bản, nhưng Phần 1.3 cung cấp kinh nghiệm xây dựng & sử dụng các quan hệ đệ quy. Các quan hệ đệ quy & quy nạp thường cung cấp cú đấm 1-2. Bạn quan tâm đến 1 chuỗi các số nguyên – có thể là 1 chuỗi phát sinh từ 1 bài toán đếm – vì vậy trước tiên hãy tìm 1 quan hệ đệ quy cho chuỗi, sau đó bạn sử dụng nó để tạo dữ liệu, & cuối cùng sử dụng quy nạp để chứng minh bất kỳ mẫu nào bạn tìm thấy. Nếu bạn đã có kinh nghiệm trước đó với quy nạp & quan hệ đệ quy, thì bạn nên thử 1 số bài toán & chuyển nhanh sang các chương tiếp theo. Tuy nhiên, việc tích lũy kinh nghiệm thiết lập các quan hệ đệ quy bằng cách giải các bài toán – có thể đồng thời khi bạn giải các chương sau – là điều rất được khuyến khích. Phần tùy chọn 1.4 giới thiệu 2 phương pháp có thể sử dụng để giải quyết các quan hệ hồi quy.

  - 1.1. Induction.
- 2. Pigeonhole Principle & Ramsey Theory.
- 3. Counting, Probability, Balls, & Boxes.
- 4. Permutations & Combinations.
- 5. Binomial & Multinomial Coefficients.
- 6. Stirling Numbers.
- 7. Integer Partitions.
- 8. Inclusion–Exclusion Principle.
- 9. Generating Functions.
- 10. Graph Theory.
- 11. Posets, Matchings, & Boolean Lattices.

2 Graph Theory – Lý Thuyết Đồ Thị

2.1 [Val02; Val21]. Gabriel Valiente. Algorithms on Trees & Graphs With Python Code

- Preface to 2e. 1e has been extensively used for graduate teaching & research all over world in last 2 decades. In this new edition, have substituted detail pseudocode for both literate programming description & implementation of algorithms using LEDA library of efficient data structures & algorithms. Although pseudocode is detailed enough to allow for a straightforward implementation of algorithms in any modern programming language, have added a proof-of-concept implementation in Python of all algorithms in Appendix A. This is, therefore, a thoroughly revised & extended edition.

  Regarding new material, have added an adjacency map representation of trees & graphs, & both maximum cardinality & maximum weight bipartite matching as an additional application of graph traversal techniques. Further, have revised end-of-chap problems & exercises & have included solutions to all problems in Appendix B.

- Preface to 1e. Graph algorithms, a long-established subject in mathematics & CS curricula, are also of much interest to disciplines e.g. computational molecular biology & computational chemistry. This book goes beyond *classical* graph problems of shortest paths, spanning trees, flows in networks, & matchings in bipartite graphs, & addresses further algorithmic problems of practical application on trees & graphs. Much of material presented on book is only available in specialized research literature.

  – Thuật toán đồ thị, 1 môn học lâu đời trong chương trình giảng dạy toán học & CS, cũng rất được các ngành học quan tâm, ví dụ như sinh học phân tử tính toán & hóa học tính toán. Cuốn sách này đi xa hơn các bài toán đồ thị *cổ điển* về đường đi ngắn nhất, cây bao trùm, luồng trong mạng, & phép ghép trong đồ thị hai phần, & giải quyết các bài toán thuật toán khác về ứng dụng thực tế trên cây & đồ thị. Phần lớn tài liệu trình bày trong sách chỉ có trong tài liệu nghiên cứu chuyên ngành.

  Book is structured around fundamental problem of isomorphism. Tree isomorphism is covered in much detail, together with related problems of subtree isomorphism, maximum common subtree isomorphism, & tree comparison. Graph isomorphism is also covered in much detail, together with related problems of subgraph isomorphism, maximum common subgraph isomorphism, & graph edit distance. A building block for solving some of these isomorphism problems are algorithms for finding maximal & maximum cliques.

  – Sách được cấu trúc xung quanh vấn đề cơ bản về phép đồng cấu. Phép đồng cấu cây được trình bày chi tiết, cùng với các vấn đề liên quan đến phép đồng cấu cây con, phép đồng cấu cây con chung cực đại, & so sánh cây. Phép đồng cấu đồ thị cũng được trình bày chi tiết, cùng với các vấn đề liên quan đến phép đồng cấu đồ thị con, phép đồng cấu đồ thị con chung cực đại, & khoảng cách chỉnh sửa đồ thị. Một khối xây dựng để giải quyết 1 số vấn đề về phép đồng cấu này là các thuật toán để tìm các nhóm & cực đại.

  Most intractable graph problems of practical application are not even approximable to within a constant bound, & several of isomorphism problems addressed in this book are no exception. Book can thus be seen as a companion to recent texts on approximation algorithms [1, 16], but also as a complement to previous texts on combinatorial & graph algorithms [2–15, 17].

  – Hầu hết các bài toán đồ thị khó giải của ứng dụng thực tế thậm chí không thể xấp xỉ trong 1 giới hạn hằng số, & 1 số bài toán đồng cấu được đề cập trong cuốn sách này cũng không ngoại lệ. Do đó, cuốn sách có thể được coi là 1 phần bổ sung cho các văn bản gần đây về thuật toán xấp xỉ [1, 16], nhưng cũng là phần bổ sung cho các văn bản trước đó về thuật toán đồ thị & tổ hợp [2–15, 17].

  Book is conceived on ground of 1st, introducing simple algorithms for these problems in order to develop some intuition before moving on to more complicated algorithms from research literature & 2nd, stimulating graduate research on tree & graph algorithms by providing together with underlying theory, a solid basis for experimentation & further development.

  – Cuốn sách được hình thành trên cơ sở thứ nhất, giới thiệu các thuật toán đơn giản cho các bài toán này để phát triển trực giác trước khi chuyển sang các thuật toán phức tạp hơn từ tài liệu nghiên cứu & Thứ hai, kích thích nghiên cứu sau đại học về thuật toán cây & đồ thị bằng cách cung cấp cùng với lý thuyết cơ bản, 1 cơ sở vững chắc cho thử nghiệm & phát triển hơn nữa.

  Algorithms are presented on an intuitive basis, followed by a detailed exposition in a literate programming style. Correctness proofs are also given, together with a worst-case analysis of algorithms. Further, full C++ implementation of all algorithms using LEDA library of efficient data structures & algorithms is given along book. These implementations include result checking of implementation correctness using correctness certificates.

  – Thuật toán được trình bày theo cách trực quan, tiếp theo là phần trình bày chi tiết theo phong cách lập trình dễ hiểu. Các bằng chứng về tính đúng đắn cũng được đưa ra, cùng với phân tích trường hợp xấu nhất của thuật toán. Ngoài ra, triển khai C++ đầy đủ của tất cả các thuật toán sử dụng thư viện LEDA về các cấu trúc dữ liệu hiệu quả & thuật toán được đưa ra cùng với sách. Các triển khai này bao gồm kiểm tra kết quả về tính đúng đắn của triển khai bằng cách sử dụng chứng chỉ tính đúng đắn.

  Choice of LEDA, which is becoming a de-facto standard for graduate courses on graph algorithms throughout world is not casual, because it allows student, lecturer, researcher, & practitioner to complement algorithmic graph theory with actual implementation & experimentation, building upon a thorough library of efficient implementations of modern data structures & fundamental algorithms.

  – Việc lựa chọn LEDA, đang trở thành tiêu chuẩn thực tế cho các khóa học sau đại học về thuật toán đồ thị trên toàn thế giới, không phải là việc tùy tiện, vì nó cho phép sinh viên, giảng viên, nhà nghiên cứu, & người thực hành bổ sung lý thuyết

đồ thị thuật toán bằng cách triển khai & thử nghiệm thực tế, dựa trên 1 thư viện toàn diện về các triển khai hiệu quả của các cấu trúc dữ liệu hiện đại & thuật toán cơ bản.

An interactive demonstration including animations of all algorithms using LEDA is given in an appendix. Interactive demonstration also includes visual checkers of implementation correctness.

– Một bản trình diễn tương tác bao gồm hoạt ảnh của tất cả các thuật toán sử dụng LEDA được đưa ra trong phần phụ lục. Bản trình diễn tương tác cũng bao gồm các công cụ kiểm tra trực quan về tính chính xác của việc triển khai.

Book is divided into 4 parts. Part I has an introductory nature & consists of 2 chaps. Chap. 1 includes a review of basic graph-theoretical notions & results used along book, a brief primer of literate programming, & an exposition of implementation correctness approach by result checking using correctness certificates. Chap. 2 is devoted exclusively to fundamental algorithmic techniques used in book: backtracking, branch-&-bound, divide-&-conquer, & DP. These techniques are illustrated by means of a running example: algorithms for tree edit distance problem.

– Sách được chia thành 4 phần. Phần I có tính chất giới thiệu & gồm 2 chương. Chương 1 bao gồm phần tổng quan về các khái niệm cơ bản về lý thuyết đồ thị & kết quả được sử dụng trong sách, 1 bài tóm tắt ngắn gọn về lập trình có hiểu biết, & trình bày về cách tiếp cận tính đúng đắn của việc triển khai bằng cách kiểm tra kết quả bằng cách sử dụng các chứng chỉ tính đúng đắn. Chương 2 dành riêng cho các kỹ thuật thuật toán cơ bản được sử dụng trong sách: quay lui, nhánh-&-bound, chia-&-chinh phục, & DP. Các kỹ thuật này được minh họa bằng 1 ví dụ đang chạy: các thuật toán cho bài toán khoảng cách chỉnh sửa cây.

Part II also consists of 2 chaps. Chap. 3 addresses most common methods for traversing general, rooted trees: depth-1st prefix leftmost (preorder), depth-1st prefix rightmost, depth-1st postfix leftmost (postorder), depth-1st postfix rightbost, breadth-1st leftmost (top-down), breadth-1st rightmost, & bottom-up traversal. Tree drawing is also discussed as an application of tree traversal methods. Chap. 4 addresses several isomorphism problems on ordered & unordered trees: tree isomorphism, subtree isomorphism, & maximum common subtree isomorphism. Computational molecular biology is also discussed as an application of different isomorphism problems on trees.

– Phần II cũng bao gồm 2 chương. Chương 3 đề cập đến các phương pháp phổ biến nhất để duyệt các cây có gốc chung: tiền tố độ sâu 1 bên trái nhất (thứ tự trước), tiền tố độ sâu 1 bên phải nhất, hậu tố độ sâu 1 bên trái nhất (thứ tự sau), hậu tố độ sâu 1 bên phải nhất, chiều rộng 1 bên trái nhất (từ trên xuống), chiều rộng 1 bên phải nhất, duyệt & từ dưới lên. Vẽ cây cũng được thảo luận như 1 ứng dụng của các phương pháp duyệt cây. Chương 4 đề cập đến 1 số vấn đề đồng cấu trên cây có thứ tự & không có thứ tự: đồng cấu cây, đồng cấu cây con, đồng cấu cây con chung & lớn nhất. Sinh học phân tử tính toán cũng được thảo luận như 1 ứng dụng của các vấn đề đồng cấu khác nhau trên cây.

Part III consists of 3 chaps. Chap. 5 addresses most common methods for traversing graphs: depth-1st & breadth-1st traversal, which resp. generalize depth-1st prefix leftmost (preorder) & breadth-1st left-most (top-down) tree traversal. Leftmost depth-1st traversal of an undirected graph, a particular case of depth-1st traversal, is also discussed. Isomorphism of ordered graphs is also discussed as an application of graph traversal methods. Chap. 6 addresses related problems of finding cliques, independent sets, & vertex covers in trees & graphs. Multiple alignment of protein sequences in computational molecular biology is also discussed as an application of clique algorithms. Chap. 7 addresses several isomorphism problems on graphs: graph isomorphism, graph automorphism, subgraph isomorphism, & maximum common subgraph isomorphism. Chemical structure search is also discussed as an application of different graph isomorphism problems.

– Phần III gồm 3 chương. Chương 5 đề cập đến các phương pháp phổ biến nhất để duyệt đồ thị: duyệt theo chiều sâu 1 & theo chiều rộng 1, tương ứng là tổng quát hóa tiền tố chiều sâu 1 bên trái nhất (thứ tự trước) & duyệt cây theo chiều rộng 1 bên trái nhất (từ trên xuống). Duyệt theo chiều sâu 1 bên trái nhất của đồ thị vô hướng, 1 trường hợp cụ thể của duyệt theo chiều sâu 1, cũng được thảo luận. Đồng cấu của đồ thị có thứ tự cũng được thảo luận như 1 ứng dụng của các phương pháp duyệt đồ thị. Chương 6 đề cập đến các vấn đề liên quan đến việc tìm clique, các tập độc lập, & các lớp phủ đỉnh trong cây & đồ thị. Căn chỉnh nhiều chuỗi protein trong sinh học phân tử tính toán cũng được thảo luận như 1 ứng dụng của các thuật toán clique. Chương 7 đề cập đến 1 số vấn đề đồng cấu trên đồ thị: đồng cấu đồ thị, tự động cấu đồ thị, đồng cấu đồ thị con, đồng cấu & đồ thị con chung lớn nhất. Tìm kiếm cấu trúc hóa học cũng được thảo luận như 1 ứng dụng của các vấn đề đồng cấu đồ thị khác nhau.

Part IV consists of 2 appendices, followed by bibliographies references & an index. Appendix A gives an overview of LEDA, including a simple C++ representation of trees as LEDA graphs, & a C++ implementation of radix sort using LEDA. Interactive demonstration of graph algorithms presented along book is put together in Appendix B. Finally, Appendix C contains a complete index to all program modules described in book.

– Phần IV gồm 2 phụ lục, tiếp theo là các tài liệu tham khảo & 1 chỉ mục. Phụ lục A cung cấp tổng quan về LEDA, bao gồm 1 biểu diễn C++ đơn giản của cây dưới dạng đồ thị LEDA, & 1 triển khai C++ của thuật toán sắp xếp radix sử dụng LEDA. Bản trình bày tương tác về các thuật toán đồ thị được trình bày dọc theo sách được tập hợp trong Phụ lục B. Cuối cùng, Phụ lục C chứa 1 chỉ mục đầy đủ cho tất cả các mô-đun chương trình được mô tả trong sách.

This book is suitable for use in upper undergraduate & graduate level courses on algorithmic graph theory. This book can also be used as a supplementary text in basic undergraduate & graduate level courses on algorithms & data structures, & in computational molecular biology & computational chemistry courses as well. Some basic knowledge of discrete mathematics, data structures, algorithms, & programming at undergraduate level is assumed.

PART I: INTRODUCTION.

- 1. Introduction.
  - 1.1. Trees & Graphs. Notion of graph which is most useful in CS is that of a directed or just a graph. A graph is a combinatorial structure consisting of a finite nonempty set of objects, called *vertices*, together with a finite (possibly empty) set of ordered pairs of vertices, called *directed edges* or *arcs*.
  - 1.2. Basic Data Structures.
  - 1.3. Representation of Trees & Graphs.
  - Summary.
- 2. Algorithmic Techniques.

  PART II: ALGORITHMS ON TREES.
- 3. Tree Traversal.
- 4. Tree Isomorphism.

  PART III: ALGORITHMS ON GRAPHS.
- 5. Graph Traversal.
- 6. Clique, Independent Set, & Vertex Cover.
- 7. Graph Isomorphism.
- A: Implementation of Algorithms in Python.
- B: Solutions to All Problems.

3 Optimization on Graphs

3.1 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning

- Abstract. Present a framework to tackle combinatorial optimization problems using neural networks & reinforcement learning. Focus on traveling salesman problem (TSP) & train a RNN that, given a set of city coordinates, predicts a distribution over different city permutations. Using negative tour length as the reward signal, optimize parameters of RNN using a policy gradient method. Compare learning network parameters on a set of training graphs against learning them on individual test graphs. Despite computational expense, without much engineering & heuristic designing, Neural Combinatorial Optimization achieves close to optimal results on 2D Euclidean graphs with up to 100 nodes. Applied to KnapSack, another NP-hard problem, same method obtains optimal solutions e.g. with up to 200 items.

  – Trình bày 1 khuôn khổ để giải quyết các vấn đề tối ưu hóa tổ hợp bằng cách sử dụng mạng nơ-ron & học tăng cường. Tập trung vào bài toán người bán hàng du lịch (TSP) & đào tạo 1 RNN, với 1 tập hợp các tọa độ thành phố, dự đoán phân phối trên các hoán vị thành phố khác nhau. Sử dụng độ dài chuyến tham quan âm làm tín hiệu phần thưởng, tối ưu hóa các tham số của RNN bằng phương pháp gradient chính sách. So sánh các tham số mạng học trên 1 tập hợp các đồ thị đào tạo với việc học chúng trên các đồ thị thử nghiệm riêng lẻ. Mặc dù tốn kém về mặt tính toán, không cần nhiều kỹ thuật & thiết kế theo phương pháp tìm kiếm, Tối ưu hóa tổ hợp nơ-ron đạt được kết quả gần như tối ưu trên đồ thị Euclid 2D với tối đa 100 nút. Áp dụng cho KnapSack, 1 bài toán NP-khó khác, phương pháp tương tự thu được các giải pháp tối ưu, ví dụ với tối đa 200 mục.
- 1. Introduction. *Combinatorial optimization* is a fundamental problem in CS. A canonical example is *traveling salesman problem* (TSP), where given a graph, one needs to search space of permutations to find an optimal sequence of nodes with minimal total edge weights (tour length). TSP & its variants have myriad applications in planning, manufacturing, genetics, etc. (see (Applegate et al., 2011) for an overview).

  –*Tối ưu hóa tổ hợp* là 1 bài toán cơ bản trong CS. Một ví dụ điển hình là *Bài toán người bán hàng du lịch* (TSP), trong đó với 1 đồ thị, người ta cần tìm kiếm không gian hoán vị để tìm ra 1 chuỗi tối ưu các nút có tổng trọng số cạnh (chiều dài hành trình) nhỏ nhất. TSP & các biến thể của nó có vô số ứng dụng trong lập kế hoạch, sản xuất, di truyền, v.v.

  Finding optimal TSP solution is NP-hard, even in 2D Euclidean case (Papadimitriou, 1977), where nodes are 2D points & edge weights are Euclidean distances between pairs of points. In practice, TSP solvers rely on handcrafted heuristics that guide their search procedures to find competitive tours efficiently. Even though these heuristics work well on TSP, once problem statement changes slightly, they need to be revised. In contrast, ML methods have potential to be applicable across many optimization tasks by automatically discovering their own heuristics based on training data, thus requiring less hand-engineering than solvers that are optimized for 1 task only.

  – Việc tìm ra giải pháp TSP tối ưu là NP-khó, ngay cả trong trường hợp Euclidean 2D (Papadimitriou, 1977), trong đó các nút là các điểm 2D & trọng số cạnh là khoảng cách Euclidean giữa các cặp điểm. Trong thực tế, các trình giải TSP dựa vào các phương pháp tìm kiếm thủ công hướng dẫn các quy trình tìm kiếm của chúng để tìm các tour du lịch cạnh tranh 1 cách

hiệu quả. Mặc dù các phương pháp tìm kiếm này hoạt động tốt trên TSP, nhưng khi phát biểu bài toán thay đổi 1 chút, chúng cần được sửa đổi. Ngược lại, các phương pháp ML có tiềm năng áp dụng trên nhiều tác vụ tối ưu hóa bằng cách tự động khám phá các phương pháp tìm kiếm của riêng chúng dựa trên dữ liệu đào tạo, do đó đòi hỏi ít kỹ thuật thủ công hơn so với các trình giải được tối ưu hóa chỉ cho 1 tác vụ.

While most successful ML techniques fall into family of supervised learning, where a mapping fro training inputs to outputs is learned, supervised learning is not applicable to most combinatorial optimization problems because one does not have access to optimal labels. However, one can compare quality of a set of solutions using a verifier, & provide some reward feedbacks to a learning algorithm. Hence, follow reinforcement learning (RL) paradigm to tackle combinatorial optimization. Empirically demonstrate that, even when using optimal solutions as labeled data to optimize a supervised mapping, generalization is rather poor compared to an RL agent that explores different tours & observes their corresponding rewards.

– Trong khi hầu hết các kỹ thuật ML thành công đều thuộc nhóm học có giám sát, trong đó ánh xạ từ đầu vào đào tạo đến đầu ra được học, học có giám sát không áp dụng được cho hầu hết các vấn đề tối ưu hóa tổ hợp vì người ta không có quyền truy cập vào các nhãn tối ưu. Tuy nhiên, người ta có thể so sánh chất lượng của 1 tập hợp các giải pháp bằng cách sử dụng trình xác minh, & cung cấp 1 số phản hồi phần thưởng cho thuật toán học. Do đó, hãy làm theo mô hình học tăng cường (RL) để giải quyết tối ưu hóa tổ hợp. Chứng minh theo kinh nghiệm rằng, ngay cả khi sử dụng các giải pháp tối ưu làm dữ liệu có nhãn để tối ưu hóa ánh xạ có giám sát, thì khả năng khái quát hóa vẫn khá kém so với tác nhân RL khám phá các chuyến tham quan khác nhau & quan sát phần thưởng tương ứng của chúng.

Propose Neural Combinatorial Optimization, a framework to tackle combinatorial optimization problems using reinforcement learning & neural networks. Consider 2 approaches based on policy gradients (Williams, 1992). 1st approach, called *RL pretraining*, uses a training set to optimize a RNN that parameterizes a stochastic policy over solutions, using expected reward as objective. At test time, policy is fixed, & one performs inference by greedy decoding or sampling. 2nd approach, called *active search*, involves no pretraining. It starts from a random policy & iteratively optimizes RNN parameters on a single test instance, again using expected reward objective, while keeping track of best solution sampled during search. Find: combining RL pretraining & active search works best in practice.

– Đề xuất Tối ưu hóa tổ hợp nơ-ron, 1 khuôn khổ để giải quyết các vấn đề tối ưu hóa tổ hợp bằng cách sử dụng học tăng cường & mạng nơ-ron. Xem xét 2 cách tiếp cận dựa trên các gradient chính sách (Williams, 1992). Cách tiếp cận thứ nhất, được gọi là *RL pretraining*, sử dụng 1 tập huấn luyện để tối ưu hóa RNN tham số hóa 1 chính sách ngẫu nhiên trên các giải pháp, sử dụng phần thưởng mong đợi làm mục tiêu. Tại thời điểm kiểm tra, chính sách được cố định, & người ta thực hiện suy luận bằng cách giải mã tham lam hoặc lấy mẫu. Cách tiếp cận thứ hai, được gọi là *active search*, không liên quan đến việc đào tạo trước. Nó bắt đầu từ 1 chính sách ngẫu nhiên & tối ưu hóa lặp lại các tham số RNN trên 1 trường hợp kiểm tra duy nhất, 1 lần nữa sử dụng mục tiêu phần thưởng mong đợi, trong khi theo dõi giải pháp tốt nhất được lấy mẫu trong quá trình tìm kiếm. Tìm: kết hợp RL pretraining & active search hoạt động tốt nhất trong thực tế.

On 2D Euclidean graphs with up to 100 nodes, Neural Combinatorial Optimization significantly outperforms supervised learning approach to TSP (Vinyals et al., 2015b) & obtains close to optimal results when allowed more computation time. Illustrate its flexibility by testing same method on KnapSack problem, for which get optimal results e.g. with up to 200 items. These results give insights into how neural networks can be used as a general tool for tackling combinatorial optimization problems, especially those that are difficult to design heuristics for.

– Trên đồ thị Euclid 2D với tối đa 100 nút, Neural Combinatorial Optimization vượt trội hơn đáng kể so với phương pháp học có giám sát đối với TSP (Vinyals et al., 2015b) & đạt được kết quả gần tối ưu khi được phép có nhiều thời gian tính toán hơn. Minh họa tính linh hoạt của nó bằng cách thử nghiệm cùng phương pháp trên bài toán KnapSack, trong đó đạt được kết quả tối ưu, ví dụ với tối đa 200 mục. Những kết quả này cung cấp thông tin chi tiết về cách mạng nơ-ron có thể được sử dụng như 1 công cụ chung để giải quyết các bài toán tối ưu hóa tổ hợp, đặc biệt là những bài toán khó thiết kế phương pháp tìm kiếm.

- 2. Previous Work. TSP is a well studied combinatorial optimization problem & many exact or approximate algorithms have been proposed for both Euclidean & non-Euclidean graphs. Christofides (1976) proposes a heuristic algorithm that involves computing a minimum-spanning tree & a minimum-weight perfect matching. Algorithm has polynomial running time & returns solutions guaranteed to be within a factor of 1.5× to optimality in the metric instance of TSP.

  – TSP là 1 bài toán tối ưu hóa tổ hợp được nghiên cứu kỹ & nhiều thuật toán chính xác hoặc gần đúng đã được đề xuất cho cả đồ thị Euclid & phi Euclid. Christofides (1976) đề xuất 1 thuật toán heuristic liên quan đến việc tính toán 1 cây có khung nhỏ nhất & 1 phép ghép hoàn hảo có trọng số nhỏ nhất. Thuật toán có thời gian chạy đa thức & trả về các giải pháp được đảm bảo nằm trong hệ số $1,5\times$ để tối ưu trong trường hợp số liệu của TSP.

  Best known exact dynamic programming algorithm for TSP has a complexity of $\Theta(2^n n^2)$, making it infeasible to scale up to large instances, say with 40 points. Nevertheless, state of art TSP solvers, thanks to carefully handcrafted heuristics that describe how to navigate space of feasible solutions in an efficient manner, can solve symmetric TSP instances with thousands of nodes. Concorde (Applegate et al., 2006), widely accepted as 1 of best exact TSP solvers, make use of cutting plane algorithms (Dantzig et al., 1954; Padberg & Rinaldi, 1990; Applegate et al., 2003), iteratively solving linear programming relaxations of TSP, in conjunction with a branch-&-bound approach that prunes parts of search space that provably will not contain an optimal solution. Similarly, Lin-Kernighan-Helsgaun heuristic (Helsgaun, 2000), inspired from Lin-Kernighan heuristic (Lin & Kernighan, 1973), is a state of art approximate search heuristic for symmetric TSP & has been shown to solve instances with hundreds of nodes to optimality.

  – Thuật toán lập trình động chính xác nổi tiếng nhất cho TSP có độ phức tạp là $\Theta(2^n n^2)$, khiến việc mở rộng lên các trường hợp lớn, chẳng hạn như 40 điểm là không khả thi. Tuy nhiên, các trình giải TSP hiện đại, nhờ vào các thuật toán tìm kiếm

được chế tạo thủ công cẩn thận mô tả cách điều hướng không gian các giải pháp khả thi theo cách hiệu quả, có thể giải các trường hợp TSP đối xứng với hàng nghìn nút. Concorde (Applegate và cộng sự, 2006), được chấp nhận rộng rãi là 1 trong những trình giải TSP chính xác tốt nhất, sử dụng các thuật toán mặt phẳng cắt (Dantzig và cộng sự, 1954; Padberg & Rinaldi, 1990; Applegate và cộng sự, 2003), giải quyết lặp đi lặp lại các phép nới lỏng lập trình tuyến tính của TSP, kết hợp với phương pháp tiếp cận nhánh&-bound cắt tỉa các phần của không gian tìm kiếm có thể chứng minh được là không chứa giải pháp tối ưu. Tương tự như vậy, phương pháp tìm kiếm gần đúng Lin-Kernighan-Helsgaun (Helsgaun, 2000), lấy cảm hứng từ phương pháp tìm kiếm gần đúng Lin-Kernighan (Lin & Kernighan, 1973), là 1 phương pháp tìm kiếm gần đúng tiên tiến cho TSP đối xứng & đã được chứng minh là có thể giải quyết các trường hợp có hàng trăm nút đến mức tối ưu.

More generic solvers, e.g. Google's vehicle routing problem solver (Google, 2016) that tackles a superset of TSP, typically rely on a combination of local search algorithms & metaheuristics. Local search algorithms apply a specified set of local move operators on candidate solutions, based on hand-engineered heuristics e.g. 2-opt (Johnson, 1990), to navigate from solution to solution in search space. A metaheuristic is then applied to propose uphill moves & escape local optima. A popular choice of metaheuristic for TSP & its variants is guided local search (Voudouris & Tsang, 1999), which moves out of a local minimum by penalizing particular solution features that it considers should not occur in a good solution.

– Các trình giải chung hơn, ví dụ như trình giải bài toán định tuyến xe của Google (Google, 2016) giải quyết 1 siêu tập hợp TSP, thường dựa vào sự kết hợp của các thuật toán tìm kiếm cục bộ & siêu thuật toán. Các thuật toán tìm kiếm cục bộ áp dụng 1 tập hợp các toán tử di chuyển cục bộ được chỉ định trên các giải pháp ứng viên, dựa trên các thuật toán tìm kiếm được thiết kế thủ công, ví dụ như 2-opt (Johnson, 1990), để điều hướng từ giải pháp này sang giải pháp khác trong không gian tìm kiếm. Sau đó, 1 siêu thuật toán được áp dụng để đề xuất các bước di chuyển lên dốc & thoát khỏi tối ưu cục bộ. Một lựa chọn phổ biến của siêu thuật toán cho TSP & các biến thể của nó là tìm kiếm cục bộ có hướng dẫn (Voudouris & Tsang, 1999), di chuyển ra khỏi mức tối thiểu cục bộ bằng cách phạt các tính năng giải pháp cụ thể mà nó cho là không nên xảy ra trong 1 giải pháp tốt.

Difficulty in applying existing search heuristics to newly encountered problems – or even new instances of a similar problem – is a well-known challenge that stems from *No Free Lunch theorem* (Wolpert & Macready, 1997). Because all search algorithms have same performance when averaged over all problems, one must appropriately rely on a prior over problems when selecting a search algorithm to guarantee performance. This challenge has fostered interest in raising level of generality at which optimization systems operate (Burke et al., 2003) & is underlying motivation behind hyper-heuristics, defined as "search method[s] or learning mechanism[s] for selecting or generating heuristics to solve computation search problems". Hyper-heuristics aim to be easier to use than problem specific methods by partially abstracting away knowledge intensive process of selecting heuristics given a combinatorial problem & have been shown to successfully combine human-defined heuristics in superior ways across many tasks (see (Burke et al., 2013) for a survey). However, hyper-heuristics operate on search space of heuristics, rather than search space of solutions, therefore still initially relying on human created heuristics.

– Khó khăn trong việc áp dụng các phương pháp tìm kiếm hiện có vào các vấn đề mới gặp – hoặc thậm chí là các trường hợp mới của 1 vấn đề tương tự – là 1 thách thức nổi tiếng bắt nguồn từ *Định lý Không có bữa trưa miễn phí* (Wolpert & Macready, 1997). Bởi vì tất cả các thuật toán tìm kiếm đều có cùng hiệu suất khi tính trung bình trên tất cả các vấn đề, nên người ta phải dựa vào 1 cách thích hợp vào 1 vấn đề trước khi chọn 1 thuật toán tìm kiếm để đảm bảo hiệu suất. Thách thức này đã thúc đẩy sự quan tâm trong việc nâng cao mức độ tổng quát mà các hệ thống tối ưu hóa hoạt động (Burke và cộng sự, 2003) & là động lực cơ bản đằng sau các siêu phương pháp tìm kiếm, được định nghĩa là "phương pháp tìm kiếm hoặc cơ chế học tập để lựa chọn hoặc tạo ra các phương pháp tìm kiếm nhằm giải quyết các vấn đề tìm kiếm tính toán". Các siêu phương pháp tìm kiếm hướng đến mục đích dễ sử dụng hơn các phương pháp cụ thể của vấn đề bằng cách trừu tượng hóa 1 phần quá trình chuyên sâu về kiến thức để lựa chọn các phương pháp tìm kiếm cho 1 vấn đề kết hợp & đã được chứng minh là kết hợp thành công các phương pháp tìm kiếm do con người xác định theo những cách vượt trội trong nhiều nhiệm vụ (xem (Burke và cộng sự, 2013) để biết khảo sát). Tuy nhiên, siêu phương pháp tìm kiếm hoạt động trên không gian tìm kiếm của phương pháp tìm kiếm, thay vì không gian tìm kiếm các giải pháp, do đó ban đầu vẫn dựa trên phương pháp tìm kiếm do con người tạo ra.

Application of neural networks to combinatorial optimization has a distinguished history, where majority of research focuses on TSP (Smith, 1999). 1 of earliest proposals is use of Hopfield networks (Hopfield & Tank, 1985) for TSP. Authors modify network's energy function to make it equivalent to TSP objective & use Lagrange multipliers to penalize violations of problem's constraints. A limitation of this approach: sensitive to hyperparameters & parameter initialization as analyzed by (Wilson & Pawley, 1988). Overcoming this limitation is central to subsequent work in field, especially by (Aiyer et al., 1990; Gee, 1993). Parallel to development of Hopfield networks is work on using deformable template models to solve TSP. Perhaps most prominent is invention of Elastic Nets as a means to solve TSP (Durbin, 1987), & application of Self Organizing Map to TSP (Fort, 1988; Angeniol et al., 1988; Kohonen, 1990). Addressing limitations of deformable template models is central to following work in this area (Burke, 1994; Favata & Walker, 1991; Vakhutinsky & Golden, 1995). Even though these neural networks have many appealing properties, they are still limited as research work. When being carefully benchmarked, they are not yielded satisfying results compared to algorithmic methods (Sarwar & Bhatti, 2012; La Maire & Mladenov, 2012). Perhaps due to negative results, this research direction is largely overlooked since turn of century.

– Ứng dụng mạng nơ-ron vào tối ưu hóa tổ hợp có lịch sử đáng chú ý, trong đó phần lớn nghiên cứu tập trung vào TSP (Smith, 1999). 1 trong những đề xuất sớm nhất là sử dụng mạng Hopfield (Hopfield & Tank, 1985) cho TSP. Các tác giả sửa đổi hàm năng lượng của mạng để làm cho nó tương đương với mục tiêu TSP & sử dụng hệ số nhân Lagrange để phạt các vi phạm các ràng buộc của bài toán. Một hạn chế của phương pháp này: nhạy cảm với siêu tham số & khởi tạo tham số như đã phân tích bởi (Wilson & Pawley, 1988). Khắc phục hạn chế này là trọng tâm của công trình tiếp theo trong lĩnh vực này, đặc biệt là của (Aiyer và cộng sự, 1990; Gee, 1993). Song song với sự phát triển của mạng Hopfield là công trình sử dụng các mô

hình mẫu biến dạng để giải quyết TSP. Có lẽ nổi bật nhất là phát minh ra Elastic Nets như 1 phương tiện để giải quyết TSP (Durbin, 1987), & ứng dụng của Self Organizing Map vào TSP (Fort, 1988; Angeniol và cộng sự, 1988; Kohonen, 1990). Việc giải quyết các hạn chế của các mô hình mẫu biến dạng là trọng tâm để theo dõi công việc trong lĩnh vực này (Burke, 1994; Favata & Walker, 1991; Vakhutinsky & Golden, 1995). Mặc dù các mạng nơ-ron này có nhiều đặc tính hấp dẫn, chúng vẫn còn hạn chế như công trình nghiên cứu. Khi được đánh giá chuẩn cẩn thận, chúng không mang lại kết quả thỏa đáng so với các phương pháp thuật toán (Sarwar & Bhatti, 2012; La Maire & Mladenov, 2012). Có lẽ do kết quả tiêu cực, hướng nghiên cứu này phần lớn bị bỏ qua kể từ đầu thế kỷ.

Motivated by recent advancements in sequence-to-sequence learning (Sutskever et al., 2014), neural networks are again subject of study for optimization in various domains (Yutian et al., 2016), including discrete ones (Zoph & Le, 2016). In particular, TSP is revisited in introduction of Pointer Networks (Vinyals et al., 2015b), where a recurrent network with non-parametric softmaxes is trained in a supervised manner to predict sequence of visited cities. Despite architectural improvements, their models were trained using supervised signals given by an approximate solver.

– Được thúc đẩy bởi những tiến bộ gần đây trong việc học chuỗi-sang-chuỗi (Sutskever và cộng sự, 2014), mạng nơ-ron 1 lần nữa là chủ đề nghiên cứu để tối ưu hóa trong nhiều miền khác nhau (Yutian và cộng sự, 2016), bao gồm cả miền rời rạc (Zoph & Le, 2016). Đặc biệt, TSP được xem xét lại trong phần giới thiệu về Mạng con trỏ (Vinyals và cộng sự, 2015b), trong đó 1 mạng hồi quy với softmax không tham số được đào tạo theo cách có giám sát để dự đoán chuỗi các thành phố đã ghé thăm. Mặc dù có những cải tiến về mặt kiến trúc, các mô hình của họ vẫn được đào tạo bằng các tín hiệu có giám sát do 1 trình giải gần đúng cung cấp.

- 3. Neural Network Architecture for TSP. Focus on 2D Euclidean TSP. Given an input graph, represented as a sequence of n cities in a 2D space $s = \{\mathbf{x}_i\}_{i=1}^n$ where each $\mathbf{x}_i \in \mathbb{R}^2$, we are concerned with finding a permutation of points π, termed a *tour*, that visits each city once & has minimum total length. Define length of a tour defined by a permutation π as

$$L(\pi|s) = \|\mathbf{x}_{\pi(n)} - \mathbf{x}_{\pi(1)}\|_2 + \sum_{i=1}^{n-1} \|\mathbf{x}_{\pi(i)} - \mathbf{x}_{\pi(i+1)}\|_2,$$

where $\|\cdot\|_2$ denotes l_2 norm.

Aim: learn parameters of a stochastic policy $p(\pi|s)$ that given an input set of points s, assigns high probabilities to short tours & low probabilities to long tours. Our neural network architecture uses chain rule to factorize probability of a tour as (2)

$$p(\pi|s) = \prod_{i=1}^{n} p(\pi(i)|\pi(< i), s),$$

& then uses individual softmax modules to represent each term on RHS of (2).

Inspired by previous work (Sutskever et al., 2014) that makes use of same factorization based on chain rule to address sequence to sequence problems like machine translation. One can use a vanilla sequence to sequence model to address TSP where output vocabulary is $[n]$. However, there are 2 major issues with this approach:

1. networks trained in this fashion cannot generalize to inputs with $> n$ cities.
2. one needs to have access to groundtruth output permutations to optimize parameters with conditional log-likelihood. Address both issues in this paper.

For generalization beyond a pre-specified graph size, follow approach of (Vinyals et al., 2015b), which makes use of a set of non-parameteric softmax modules, resembling attention mechanism from (Bahdanau et al., 2015). This approach, named *pointer network*, allows model to effectively point to a specific position in input sequence rather than predicting an index value from a fixed-size vocabulary. Employ pointer network architecture, depicted in Fig. 1: A pointer network architecture introduced by (Vinyals et al., 2015b). as our policy model to parameterize $p(\pi|s)$.

– Lấy cảm hứng từ công trình trước đây (Sutskever và cộng sự, 2014) sử dụng cùng 1 phép phân tích dựa trên quy tắc chuỗi để giải quyết các vấn đề chuỗi thành chuỗi như dịch máy. Người ta có thể sử dụng mô hình chuỗi thành chuỗi vani để giải quyết TSP trong đó từ vựng đầu ra là $[n]$. Tuy nhiên, có 2 vấn đề chính với cách tiếp cận này:

1. mạng được đào tạo theo cách này không thể khái quát hóa thành các đầu vào có $> n$ thành phố.
2. người ta cần có quyền truy cập vào các hoán vị đầu ra thực tế để tối ưu hóa các tham số với log-likelihood có điều kiện. Giải quyết cả hai vấn đề trong bài báo này.

Đối với khái quát hóa vượt ra ngoài kích thước đồ thị được chỉ định trước, hãy làm theo cách tiếp cận của (Vinyals và cộng sự, 2015b), sử dụng 1 tập hợp các mô-đun softmax không tham số, giống với cơ chế chú ý từ (Bahdanau và cộng sự, 2015). Phương pháp này, được gọi là *pointer network*, cho phép mô hình trỏ hiệu quả đến 1 vị trí cụ thể trong chuỗi đầu vào thay vì dự đoán giá trị chỉ mục từ 1 từ vựng có kích thước cố định. Sử dụng kiến trúc mạng con trỏ, được mô tả trong Hình 1: Kiến trúc mạng con trỏ do (Vinyals et al., 2015b) giới thiệu làm mô hình chính sách của chúng tôi để tham số hóa $p(\pi|s)$.

  - 3.1. Architecture details. Our pointer network comprises 2 RNN modules, encoder & decoder, both of which consist of LSTM cells (Hochreiter & Schmidhuber, 1997). Encoder network reads input sequence s, 1 city at a time, & transforms it into a

sequence of latent memory state $\{\text{enc}_i\}_{i=1}^n$ where $\text{enc}_i \in \mathbb{R}^d$. Input to encoder network at time step i is a d-dimensional embedding of a 2D point $\mathbf{x}_i$, which is obtained via a linear transformation of $\mathbf{x}_i$ shared across all input steps. Decoder network also maintains its latent memory states $\{\text{dec}_i\}_{i=1}^n$ where $\text{dec}_i \in \mathbb{R}^d$ &, at each step i, uses a pointing mechanism to produce a distribution over next city to visit in tour. Once next city is selected, it is passed as input to next decoder step. Input of 1st decoder step (denoted by $\langle g \rangle$ in Fig. 1) is a d-dimensional vector treated as a trainable parameter of our neural network.

– Mạng con trỏ của chúng tôi bao gồm 2 mô-đun RNN, bộ mã hóa & bộ giải mã, cả hai đều bao gồm các ô LSTM (Hochreiter & Schmidhuber, 1997). Mạng mã hóa đọc chuỗi đầu vào s, 1 thành phố tại 1 thời điểm, & chuyển đổi nó thành 1 chuỗi trạng thái bộ nhớ tiềm ẩn $\{\text{enc}_i\}_{i=1}^n$ trong đó $\text{enc}_i \in \mathbb{R}^d$. Đầu vào của mạng mã hóa tại bước thời gian i là nhúng d chiều của điểm 2D $\mathbf{x}_i$, thu được thông qua phép biến đổi tuyến tính của $\mathbf{x}_i$ được chia sẻ trên tất cả các bước đầu vào. Mạng giải mã cũng duy trì trạng thái bộ nhớ tiềm ẩn $\{\text{dec}_i\}_{i=1}^n$ trong đó $\text{dec}_i \in \mathbb{R}^d$ &, tại mỗi bước i, sử dụng cơ chế trỏ để tạo ra phân phối trên thành phố tiếp theo để ghé thăm trong tour. Khi thành phố tiếp theo được chọn, nó được truyền làm đầu vào cho bước giải mã tiếp theo. Đầu vào của bước giải mã đầu tiên (được biểu thị bằng $\langle g \rangle$ trong Hình 1) là 1 vectơ d chiều được coi là tham số có thể đào tạo của mạng nơ-ron của chúng tôi.

Our attention function, formally defined in Appendix A.1, takes an input a query vector $q = \text{dec}_i \in \mathbb{R}^d$ & a set of reference vectors $\text{ref} = \{\text{enc}_1, \ldots, \text{enc}_k\}$ where $\text{enc}_i \in \mathbb{R}^k d$, & predicts a distribution $A(\text{ref}, q)$ over set of k references. This probability distribution represents degree to which model is pointing to reference r_i upon seeing query q.

– Hàm chú ý của chúng tôi, được định nghĩa chính thức trong Phụ lục A.1, lấy đầu vào là 1 vectơ truy vấn $q = \text{dec}_i \in \mathbb{R}^d$ & 1 tập hợp các vectơ tham chiếu $\text{ref} = \{\text{enc}_1, \ldots, \text{enc}_k\}$ trong đó $\text{enc}_i \in \mathbb{R}^k d$, & dự đoán 1 phân phối $A(\text{ref}, q)$ trên tập hợp k tham chiếu. Phân phối xác suất này biểu thị mức độ mà mô hình đang trỏ đến tham chiếu r_i khi nhìn thấy truy vấn q.

Vinyals et al. (2015a) also suggest including some additional computation steps, named *glimpses*, to aggregate contributions of different parts of input sequence, very much like (Bahdanau et al., 2015). Discuss this approach in details in Appendix A.1. In our experiments, find: utilizing 1 glimpse in pointing mechanism yields performance gains at an insignificant cost latency.

– Vinyals et al. (2015a) cũng đề xuất bao gồm một số bước tính toán bổ sung, được gọi là *glances*, để tổng hợp các đóng góp của các phần khác nhau của chuỗi đầu vào, rất giống (Bahdanau et al., 2015). Thảo luận chi tiết về cách tiếp cận này trong Phụ lục A.1. Trong các thí nghiệm của chúng tôi, hãy tìm: sử dụng 1 glance trong cơ chế trỏ mang lại hiệu suất tăng với độ trễ chi phí không đáng kể.

- 4. Optimization with policy gradients. Vinyals et al. (2015b) proposes training a pointer network using a supervised loss function comprising conditional log-likelihood, which factors into a cross entropy objective between network's output probabilities & targets provided by a TSP solver. Learning from examples in such a way is undesirable for NP-hard problems because:

  1. performance of model is tied to quality of supervised labels
  2. getting high-quality labeled data is expensive & may be infeasible for new problem statements
  3. one cares more about finding a competitive solution more than replicating results of another algorithm.

  By contrast, believe RL provides an appropriate paradigm for training neural networks for combinatorial optimization, especially because these problems have relatively simple reward mechanisms that could be even used at test time. Hence propose to use model-free policy-based RL to optimize parameters of a pointer network denoted $\boldsymbol{\theta}$. Our training objective: expected tour length which, given an input graph s, is defined as (3)

$$J(\boldsymbol{\theta}|s) = \mathbb{E}_{\pi \sim p_\theta(\cdot|s)} L(\pi|s).$$

  During training, our graphs are drawn from a distribution $\mathcal{S}$, & total training objective involves sampling from distribution of graphs, i.e., $J(\boldsymbol{\theta}) = \mathbb{E}_{s \sim \mathcal{S}} J(\boldsymbol{\theta}|s)$.

  Resort to policy gradient methods & stochastic gradient descent to optimize parameters. Gradient of (3) is formulated using well-known REINFORCE algorithm (Williams, 1992): (4)

$$\nabla_\theta J(\theta|s) = \mathbb{E}_{\pi \sim p_\theta(\cdot|s)} \left[(L(\pi|s) - b(s)) \nabla_\theta \log p_\theta(\pi|s) \right],$$

  where $b(s)$ denotes a baseline function that does not depend on π & estimates expected tour length to reduce variance of gradients.

  By drawing B i.i.d sample graphs $s_1, s_2, \ldots, s_B \sim \mathcal{S}$ & sampling a single tour per graph, i.e., $\pi_i \sim p_\theta(\cdot|s_i)$, gradient in (4) is approximated with Monte Carlo sampling as follows: (5)

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i|s_i) - b(s_i)) \nabla_\theta \log p_\theta(\pi_i|s_i).$$

  A simple & popular choice of baseline $b(s)$ is an exponential moving average of rewards obtained by network over time to account for fact: policy improves with training. While this choice of baseline proved sufficient to improve upon Christofides algorithm, it suffers from not being able to differentiable between different input graphs. In particular, optimal tour π^* for a difficult graph s may be still discouraged if $L(\pi^*|s) > b$ because b is shared across all instances in batch.

– Bằng cách vẽ đồ thị mẫu B i.i.d $s_1, s_2, \ldots, s_B \sim \mathcal{S}$ & lấy mẫu một chuyến tham quan duy nhất trên mỗi đồ thị, tức là $\pi_i \sim p_\theta(\cdot|s_i)$, gradient trong (4) được xấp xỉ bằng cách lấy mẫu Monte Carlo như sau: (5)

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i|s_i) - b(s_i)) \nabla_\theta \log p_\theta(\pi_i|s_i).$$

1 lựa chọn đơn giản & phổ biến của đường cơ sở $b(s)$ là trung bình động hàm mũ của phần thưởng thu được bởi mạng theo thời gian để tính đến thực tế: chính sách được cải thiện khi đào tạo. Trong khi lựa chọn đường cơ sở này chứng tỏ là đủ để cải thiện thuật toán Christofides, nó lại gặp vấn đề là không thể phân biệt được giữa các đồ thị đầu vào khác nhau. Đặc biệt, hành trình tối ưu π^* cho đồ thị khó s vẫn có thể không được khuyến khích nếu $L(\pi^*|s) > b$ vì b được chia sẻ trên tất cả các trường hợp trong lô.

Using a parametric baseline to estimate expected tour length $\mathbb{E}_{\pi \sim p_\theta(\cdot|s)} L(\pi|s)$ typically improves learning. Therefore, introduce an auxiliary network, called a *critic* & parameterized by θ_v, to learn expected tour length found by our current policy p_θ given an input sequence s. Critic is trained with stochastic gradient descent on a mean squared error objective between its predictions $b_{\theta_v}(s)$ & actual tour lengths sampled by most recent policy. Additional objective is formulated as

$$\mathcal{L}(\theta_v) = \frac{1}{B} \sum_{i=1}^{B} \|b_{\theta_v}(s_i) - L(\pi_i|s_i)\|_2^2.$$

– Sử dụng đường cơ sở tham số để ước tính độ dài chuyến tham quan dự kiến $\mathbb{E}_{\pi \sim p_\theta(\cdot|s)} L(\pi|s)$ thường cải thiện khả năng học tập. Do đó, hãy giới thiệu một mạng lưới phụ trợ, được gọi là *critic* & được tham số hóa bởi θ_v, để tìm hiểu độ dài chuyến tham quan dự kiến được tìm thấy bởi chính sách hiện tại của chúng tôi p_θ với một chuỗi đầu vào s. Critic được đào tạo với độ dốc ngẫu nhiên giảm dần trên mục tiêu lỗi bình phương trung bình giữa các dự đoán của nó $b_{\theta_v}(s)$ & độ dài chuyến tham quan thực tế được lấy mẫu theo chính sách gần đây nhất. Mục tiêu bổ sung được xây dựng như sau

$$\mathcal{L}(\theta_v) = \frac{1}{B} \sum_{i=1}^{B} \|b_{\theta_v}(s_i) - L(\pi_i|s_i)\|_2^2.$$

Critic's architecture for TSP. Explain how our critic maps an input sequence s into a baseline prediction $b_{\theta_v}(s)$. Our critic comprises 3 neural network modules:

1. an LSTM encoder
2. an LSTM process block
3. a 2-layer ReLU neural network decoder.

Its encoder has same architecture as that of our pointer network's encoder & encodes an input sequence s into a sequence of latent memory states & a hidden state h. Process block, similarly to (Vinyals et al., 2015a), then performs P steps of computation over hidden state h. Each processing step updates this hidden state by glimpsing at memory states as described in Appendix A.1 & feeds output of glimpse function as input to next processing step. At end of process block, obtained hidden state is then decoded into a baseline prediction (i.e. a single scalar) by 2 fully connected layers with respectively d & 1 unit(s).

– Bộ mã hóa của nó có cùng kiến trúc với bộ mã hóa mạng con trỏ của chúng tôi & mã hóa một chuỗi đầu vào s thành một chuỗi các trạng thái bộ nhớ tiềm ẩn & một trạng thái ẩn h. Khối quy trình, tương tự như (Vinyals et al., 2015a), sau đó thực hiện P bước tính toán trên trạng thái ẩn h. Mỗi bước xử lý cập nhật trạng thái ẩn này bằng cách nhìn thoáng qua các trạng thái bộ nhớ như được mô tả trong Phụ lục A.1 & đưa đầu ra của hàm nhìn thoáng qua làm đầu vào cho bước xử lý tiếp theo. Vào cuối khối quy trình, trạng thái ẩn thu được sau đó được giải mã thành một dự đoán cơ sở (tức là một số vô hướng đơn) bởi 2 lớp được kết nối đầy đủ với tương ứng d & 1 đơn vị.

Our training algorithm, described in Algorithm 1: Actor-critic training, is closely related to asynchronous advantage actor-critic (A3C) proposed in (Mnih et al., 2016), as difference between sampled tour lengths & critic's predictions is an unbiased estimate of advantage function. Perform our updates asynchronously across multiple workers, but each worker also handles a mini-batch of graphs for better gradient estimates.

– Thuật toán đào tạo của chúng tôi, được mô tả trong Thuật toán 1, có liên quan chặt chẽ đến tác nhân-phê bình lợi thế không đồng bộ (A3C) được đề xuất trong (Mnih et al., 2016), vì sự khác biệt giữa độ dài chuyến tham quan được lấy mẫu & dự đoán của nhà phê bình là ước tính không thiên vị của hàm lợi thế. Thực hiện các bản cập nhật của chúng tôi một cách không đồng bộ trên nhiều công nhân, nhưng mỗi công nhân cũng xử lý một lô nhỏ các đồ thị để ước tính độ dốc tốt hơn.

1. 4.1. Search strategies. As evaluating a tour length is inexpensive, our TSP agent can easily simulate a search procedure at inference time by considering multiple candidate solutions per graph & selecting best. This inference process resembles how solvers search over a large set of feasible solutions. In this paper, consider 2 search strategies detailed below, which refer to as *sampling & active search.*

   – *Chiến lược tìm kiếm.* Vì việc đánh giá độ dài của một chuyến tham quan không tốn kém, nên tác nhân TSP của chúng tôi có thể dễ dàng mô phỏng một quy trình tìm kiếm tại thời điểm suy luận bằng cách xem xét nhiều giải pháp ứng viên trên mỗi đồ thị & chọn giải pháp tốt nhất. Quy trình suy luận này giống với cách trình giải tìm kiếm trên một tập hợp lớn

các giải pháp khả thi. Trong bài báo này, hãy xem xét 2 chiến lược tìm kiếm được trình bày chi tiết bên dưới, được gọi là lấy mẫu & tìm kiếm chủ động.

Sampling. Our 1st approach is simply to sample multiple candidate tours from our stochastic policy $p_\theta(\cdot|s)$ & select shortest one. In contrast to heuristic solvers, do not enforce our model to sample different tours during process. However, can control diversity of sampled tours with a temperature hyperparameter when sampling from our non-parametric softmax (Appendix A.2). This sampling process yields significant improvements over greedy decoding, which always selects index with largest probability. Also considered perturbing pointing mechanism with random noise & greedily decoding from obtained modified policy, similarly to (Cho, 2016), but this proves less effective than sampling in our experiments.

– **Lấy mẫu.** Cách tiếp cận đầu tiên của chúng tôi chỉ đơn giản là lấy mẫu nhiều chuyến đi ứng viên từ chính sách ngẫu nhiên $p_\theta(\cdot|s)$ & chọn chuyến đi ngắn nhất. Trái ngược với các trình giải quyết theo phương pháp heuristic, không áp đặt mô hình của chúng tôi để lấy mẫu các chuyến đi khác nhau trong quá trình. Tuy nhiên, có thể kiểm soát tính đa dạng của các chuyến đi được lấy mẫu bằng siêu tham số nhiệt độ khi lấy mẫu từ softmax không tham số của chúng tôi (Phụ lục A.2). Quá trình lấy mẫu này mang lại những cải tiến đáng kể so với giải mã tham lam, giải mã này luôn chọn chỉ mục có xác suất lớn nhất. Cũng được xem xét cơ chế trỏ nhiễu với nhiễu ngẫu nhiên & giải mã tham lam từ chính sách đã sửa đổi thu được, tương tự như (Cho, 2016), nhưng điều này tỏ ra kém hiệu quả hơn so với lấy mẫu trong các thí nghiệm của chúng tôi.

2. **Active Search.** Rather than sampling with a fixed model & ignoring reward information obtained from sampled solutions, one can refine parameters of stochastic policy p_θ during inference to minimize $\mathbb{E}_{\pi\sim p_\theta(\cdot|s)}L(\pi|s)$ on a *single test input* s. This approach proves especially competitive when starting from a trained model. Remarkably, it also produces satisfying solutions when starting from an untrained model. Refer to these 2 approaches as *RL pretraining-Active Search & Active Search* because model actively updates its parameters while searching for candidate solutions on a single test instance.

   – **Tìm kiếm chủ động.** Thay vì lấy mẫu bằng một mô hình cố định & bỏ qua thông tin phần thưởng thu được từ các giải pháp lấy mẫu, người ta có thể tinh chỉnh các tham số của chính sách ngẫu nhiên p_θ trong quá trình suy luận để giảm thiểu $\mathbb{E}_{\pi\sim p_\theta(\cdot|s)}L(\pi|s)$ trên *đầu vào thử nghiệm đơn* s. Cách tiếp cận này tỏ ra đặc biệt cạnh tranh khi bắt đầu từ một mô hình đã được đào tạo. Đáng chú ý, nó cũng tạo ra các giải pháp thỏa mãn khi bắt đầu từ một mô hình chưa được đào tạo. Gọi 2 cách tiếp cận này là *RL pretraining-Tìm kiếm chủ động & Tìm kiếm chủ động* vì mô hình chủ động cập nhật các tham số của nó trong khi tìm kiếm các giải pháp ứng viên trên một trường hợp thử nghiệm duy nhất.

   Active Search applies policy gradients similarly to Algorithm 1 but draws Monte Carlo samples over candidate solutions $\pi_1 \dots \pi_B \sim p_\theta(\cdot|s)$ for a single test input. It resorts to an exponential moving average baseline, rather than a critic, as there is no need to differentiate between inputs. Our Active Search training algorithm is presented in Algorithm 2: Active Search. While RL training does not require supervision, it still requires training data & hence generalization depends on training data distribution. In contrast, Active Search is distribution independent. Finally, since we encode a set of cities as a sequence, randomly shuffle input sequence before feeding it to our pointer network. This increases stochasticity of sampling procedure & leads to large improvements in Active Search. Table 1: Different learning configuration.

   – Thuật toán tìm kiếm chủ động áp dụng các gradient chính sách tương tự như Thuật toán 1 nhưng rút các mẫu Monte Carlo trên các giải pháp ứng viên $\pi_1 \dots \pi_B \sim p_\theta(\cdot|s)$ cho một đầu vào thử nghiệm duy nhất. Thuật toán này sử dụng đường cơ sở trung bình động theo hàm mũ, thay vì một nhà phê bình, vì không cần phải phân biệt giữa các đầu vào. Thuật toán đào tạo Tìm kiếm chủ động của chúng tôi được trình bày trong Thuật toán 2. Trong khi đào tạo RL không yêu cầu giám sát, thì nó vẫn yêu cầu dữ liệu đào tạo & do đó, khái quát hóa phụ thuộc vào phân phối dữ liệu đào tạo. Ngược lại, Tìm kiếm chủ động không phụ thuộc vào phân phối. Cuối cùng, vì chúng tôi mã hóa một tập hợp các thành phố dưới dạng một chuỗi, nên hãy xáo trộn ngẫu nhiên chuỗi đầu vào trước khi đưa vào mạng con trỏ của chúng tôi. Điều này làm tăng tính ngẫu nhiên của quy trình lấy mẫu & dẫn đến những cải tiến lớn trong Tìm kiếm chủ động.

- 5. Experiments. Conduct experiments to investigate behavior of proposed Neural Combinatorial Optimization methods. Consider 3 benchmark tasks, Euclidean TSP20, 50 & 100, for which we generate a test set of 1000 graphs. Points are drawn uniformly at random in unit square $[0, 1]^2$.
  - 5.1. Experimental details.
  - 5.2. Results & analyses.
- 6. Generalization to other problems. In this sect, discuss how to apply Neural Combinatorial Optimization to other problems than TSP. In Neural Combinatorial Optimization, model architecture is tied to given combinatorial optimization problem. Examples of useful networks include pointer network, when output is a permutation or a truncated permutation or a subset of input, & classical `seq2seq` model for other kinds of structured outputs. For combinatorial problems that require to assign labels to elements of input, e.g. graph coloring, also possible to combine a pointer module & a softmax module to simultaneously point & assign at decoding time. Given a model that encodes an instance of a given combinatorial optimization task & repeatedly branches into subtrees to construct a solution, training procedures described in Sect. 4 can then be applied by adapting reward function depending on optimization problem being considered.

  – *Tổng quát hóa cho các vấn đề khác.* Trong phần này, thảo luận về cách áp dụng Tối ưu hóa tổ hợp nơ-ron cho các vấn đề khác ngoài TSP. Trong Tối ưu hóa tổ hợp nơ-ron, kiến trúc mô hình được gắn với bài toán tối ưu hóa tổ hợp đã cho. Các ví dụ về mạng hữu ích bao gồm mạng con trỏ, khi đầu ra là một hoán vị hoặc một hoán vị bị cắt cụt hoặc một tập hợp con của đầu vào, & mô hình `seq2seq` cổ điển cho các loại đầu ra có cấu trúc khác. Đối với các bài toán tổ hợp yêu cầu gán nhãn cho các phần tử đầu vào, ví dụ như tô màu đồ thị, cũng có thể kết hợp một mô-đun con trỏ & một mô-đun softmax để trỏ đồng

thời & gán tại thời điểm giải mã. Với một mô hình mã hóa một thể hiện của một tác vụ tối ưu hóa tổ hợp đã cho & nhiều lần phân nhánh thành các cây con để xây dựng một giải pháp, các quy trình đào tạo được mô tả trong Phần 4 sau đó có thể được áp dụng bằng cách điều chỉnh hàm phần thưởng tùy thuộc vào bài toán tối ưu hóa đang được xem xét.

Additionally, one also needs to ensure feasibility of obtained functions. For certain combinatorial problems, straightforward to know exactly which branches do not lead to any feasible solutions at decoding time. Can then simply manually assign them a 0 probability when decoding, similarly to how we enforce our model to not point at a same city twice in our pointing mechanism (Appendix A.1). However, for many combinatorial problems, coming up with a feasible solution can be a challenge in itself. Consider, e.g., TSP with Time Windows, where traveling salesman has additional constraint of visiting each city during a specific time window. It might be that most branches being considered early in tour do not lead to any solution that respects all time windows. In such cases, knowing exactly which branches are feasible requires researching their subtrees, a time-consuming process that is not much easier than directly searching for optimal solution unless using problem-specific heuristics.

– Ngoài ra, người ta cũng cần đảm bảo tính khả thi của các hàm thu được. Đối với một số bài toán tổ hợp, dễ dàng biết chính xác nhánh nào không dẫn đến bất kỳ giải pháp khả thi nào tại thời điểm giải mã. Sau đó, có thể chỉ cần gán thủ công cho chúng một xác suất 0 khi giải mã, tương tự như cách chúng ta thực thi mô hình của mình để không trỏ đến cùng một thành phố hai lần trong cơ chế trỏ của mình (Phụ lục A.1). Tuy nhiên, đối với nhiều bài toán tổ hợp, việc đưa ra một giải pháp khả thi có thể là một thách thức. Ví dụ, hãy xem xét TSP với Cửa sổ thời gian, trong đó nhân viên bán hàng du lịch có thêm ràng buộc là phải đến thăm từng thành phố trong một cửa sổ thời gian cụ thể. Có thể là hầu hết các nhánh được xem xét sớm trong chuyến đi không dẫn đến bất kỳ giải pháp nào tôn trọng tất cả các cửa sổ thời gian. Trong những trường hợp như vậy, việc biết chính xác nhánh nào khả thi đòi hỏi phải nghiên cứu các cây con của chúng, một quá trình tốn thời gian không dễ hơn nhiều so với việc tìm kiếm trực tiếp giải pháp tối ưu trừ khi sử dụng các phương pháp tìm kiếm cụ thể cho từng vấn đề.

Rather than explicitly constraining model to only sample feasible solutions, one can also let model learn to respect problem's constraints. A simple approach, to be verified experimentally in future work, consists in augmenting objective function with a term that penalizes solutions for violating problem's constraints, similarly to penalty methods in constrained optimization. While this does not guarantee model consistently samples feasible solutions at inference time, this is not necessarily problematic as we can simply ignore infeasible solutions & resample from model (for RL pretraining-Sampling & RL-pretraining Active Search). Also conceivable to combine both approaches by assigning 0 probabilities to branches that are easily identifiable as infeasible while still penalizing infeasible solutions once they are entirely constructed.

– Thay vì ràng buộc rõ ràng mô hình chỉ lấy mẫu các giải pháp khả thi, người ta cũng có thể để mô hình học cách tôn trọng các ràng buộc của vấn đề. Một cách tiếp cận đơn giản, sẽ được xác minh bằng thực nghiệm trong công việc trong tương lai, bao gồm việc tăng cường hàm mục tiêu bằng một thuật ngữ phạt các giải pháp vi phạm các ràng buộc của vấn đề, tương tự như các phương pháp phạt trong tối ưu hóa bị ràng buộc. Mặc dù điều này không đảm bảo mô hình luôn lấy mẫu các giải pháp khả thi tại thời điểm suy luận, nhưng điều này không nhất thiết là vấn đề vì chúng ta có thể chỉ cần bỏ qua các giải pháp không khả thi & lấy mẫu lại từ mô hình (đối với RL pretraining-Sampling & RL-pretraining Active Search). Cũng có thể kết hợp cả hai cách tiếp cận bằng cách gán 0 xác suất cho các nhánh dễ dàng xác định là không khả thi trong khi vẫn phạt các giải pháp không khả thi sau khi chúng được xây dựng hoàn toàn.

1. 6.1. KnapSack example. As an example of flexibility of Neural Combinatorial Optimization, consider KnapSack problem, another intensively studied problem in CS. Given a set of n items $i \in [n]$, each with weight w_i & value v_i & a maximum weight capacity of W, 0-1 KnapSack problem consists in maximizing sum of values of items present in knapsack so that sum of weights $\leq$ knapsack capacity:

$$\max_{S \subseteq [n]} \sum_{i \in S} v_i \text{ subject to } \sum_{i \in S} w_i \leq W.$$

With w_i, v_i, W taking real values, problem is NP-hard (Kellerer et al., 2004). A simple yet strong heuristic: take items ordered by their weight-to-value ratios until they fill up weight capacity. Apply pointer network & encode each knapsack instance as a sequence of 2D vectors (w_i, v_i). At decoding time, pointer network points to items to include in knapsack & stops when total weight of items collected so far exceeds weight capacity.

– *Ví dụ về KnapSack.* Là một ví dụ về tính linh hoạt của Tối ưu hóa tổ hợp nơ-ron, hãy xem xét bài toán KnapSack, một bài toán khác được nghiên cứu chuyên sâu trong CS. Cho một tập hợp n mục $i \in [n]$, mỗi mục có trọng số w_i & giá trị v_i & sức chứa trọng số tối đa là W, 0-1 Bài toán KnapSack bao gồm việc tối đa hóa tổng các giá trị của các mục có trong ba lô sao cho tổng các trọng số $\leq$ sức chứa ba lô:

$$\max_{S \subseteq [n]} \sum_{i \in S} v_i \text{ tùy thuộc vào } \sum_{i \in S} w_i \leq W.$$

Với w_i, v_i, W lấy các giá trị thực, bài toán là NP-khó (Kellerer et al., 2004). Một phương pháp tìm kiếm đơn giản nhưng mạnh mẽ: lấy các mục được sắp xếp theo tỷ lệ trọng lượng trên giá trị của chúng cho đến khi chúng lấp đầy sức chứa trọng lượng. Áp dụng mạng con trỏ & mã hóa từng trường hợp ba lô dưới dạng một chuỗi các vectơ 2D (w_i, v_i). Tại thời điểm giải mã, mạng con trỏ trỏ đến các mục để đưa vào ba lô & dừng lại khi tổng trọng lượng của các mục được thu thập cho đến nay vượt quá sức chứa trọng lượng.

Generate 3 datasets, KNAP50, KNAP100 & KNAP200, of 1000 instances with items' weights & values drawn uniformly at random in $[0, 1]$. W.l.o.g. (since we can scale items' weights), set capacities to 12.5 for KNAP50 & 25 for KNAP100 &

KNAP200. Present performance of RL pretraining-Greedy & Active Search (which we run for 5000 training steps) in Table 5: Results of RL pretraining-Greedy & Active Search on KnapSack (higher is better). & compare them to 2 simple baselines: 1st baseline is greedy weight-to-value ratio heuristic; 2nd baseline is random search, where we sample as many feasible solutions as seen by Active Search. RL pretraining-Greedy yields solutions that, in average, are just 1% less than optimal & Active Search solves all instances to optimality.

– Tạo 3 tập dữ liệu, KNAP50, KNAP100 & KNAP200, gồm 1000 trường hợp với trọng số của các mục & giá trị được rút ra ngẫu nhiên đồng đều trong $[0, 1]$. W.l.o.g. (vì chúng ta có thể chia tỷ lệ trọng số của các mục), đặt dung lượng thành $12,5$ cho KNAP50 & 25 cho KNAP100 & KNAP200. Hiệu suất hiện tại của RL pretraining-Greedy & Active Search (mà chúng ta chạy trong 5000 bước đào tạo) trong Bảng 5: Kết quả của RL pretraining-Greedy & Active Search trên KnapSack (cao hơn là tốt hơn). & so sánh chúng với 2 đường cơ sở đơn giản: Đường cơ sở thứ nhất là phương pháp tìm kiếm theo tỷ lệ trọng số-giá trị tham lam; Đường cơ sở thứ hai là tìm kiếm ngẫu nhiên, trong đó chúng ta lấy mẫu nhiều giải pháp khả thi nhất mà Active Search thấy. Tiền đào tạo RL - Greedy tạo ra các giải pháp trung bình chỉ kém tối ưu 1% & Tìm kiếm chủ động giải quyết mọi trường hợp theo hướng tối ưu.

- 7. Conclusion. This paper presents Neural Combinatorial Optimization, a framework to tackle combinatorial optimization with reinforcement learning & neural networks. Focus on TSP & present a set of results for each variation of framework. Experiments demonstrate: Neural Combinatorial Optimization achieves close to optimal results on 2D Euclidean graphs with up to 100 nodes.

  – Bài báo này trình bày về Neural Combinatorial Optimization, một khuôn khổ để giải quyết tối ưu hóa tổ hợp với học tăng cường & mạng nơ-ron. Tập trung vào TSP & trình bày một tập hợp kết quả cho từng biến thể của khuôn khổ. Các thí nghiệm chứng minh: Neural Combinatorial Optimization đạt được kết quả gần như tối ưu trên đồ thị Euclid 2D với tối đa 100 nút.

3.2 [Gol18]. Boris Goldengorin. Optimization Problems in Graph Theory. 2018

- Preface. Problem raised by Moon in 1960s, of estimating minimum possible number of vertices in a tournament that contains every k-vertex tournament as an induced subgraph. Main result: this number is $(1 + o(1))2^{\frac{k-1}{2}}$, improving earlier estimates of several researchers. Proof combines combinatorial & probabilistic techniques with group theoretic tools.

4 Wikipedia's

4.1 Wikipedia/extremal combinatorics

"*Extremal combinatorics* is a field of mathematics, which is itself a part of mathematics. Extremal combinatorics studies how large or how small a collection of finite objects (numbers, graphs, vectors, sets, etc.) can be, if it has to satisfy certain restrictions.

Much of extremal combinatorics concerns classes of sets; this is called *extremal set theory*. E.g., in an n-element set, what is the largest number of k-element subsets that can pairwise intersect one another? What is the largest number of subsets of which more contains any other? The latter question is answered by Sperner's theorem, which gave rise to much of extremal set theory.

Another kind of example: How many people can be invited to a party where among each 3 people there are 2 who know each other & 2 who don't know each other? Ramsey theory shows: at most 5 persons can attend such a party (see Theorem on Friends & Strangers). Or, suppose given a finite set of nonzero integers, & are asked to mark as large a subset as possible of this set under the restriction that the sum of any 2 marked integers cannot be marked. It appears that (independent of what the given integers actually are) we can always mark at least $\frac{1}{3}$ of them." – Wikipedia/extremal combinatorics

4.2 Wikipedia/extremal graph theory

"Turán graph $T(n, r)$ is an example of an extremal graph. It has the maximum possible number of edges for a graph on n vertices without $(r+1)$-cliques. This is $T(13, 4)$. *Extremal graph theory* is a branch of combinatorics, itself an area of mathematics, that lies at the intersection of extremal combinatorics & graph theory. In essence, extremal graph theory studies how global properties of a graph influence local substructure. Results in extremal graph theory deal with quantitative connections between various graph properties, both global (e.g. number of vertices & edges) & local (e.g. existence of specific subgraphs), & problems in extremal graph theory can often be formulated as optimization problems: how big or small a parameter of a graph can be, given some constraints that the graph has to satisfy? A graph that is an optimal solution to such an optimization problem is called an *extremal graph*, & extremal graphs are important objects of study in extremal graph theory.

Extremal graph theory is closely related to fields e.g. Ramsey theory, spectral graph theory, computational complexity theory, & additive combinatorics, & frequently employs probabilistic method.

4.2.1 History

> "Extremal graph theory, in its strictest sense, is a branch of graph theory developed & loved by Hungarians." – Bollobás (2004)

Mantel's Theorem (1907) & Turán's Theorem (1941) were some of 1st milestones in stud of extremal graph theory. In particular, Turán's theorem would later on become a motivation for the finding of results e.g. Erdős–Stone theorem (1946). This result is surprising because it connects chromatic number with maximal number of edges in an H-free graph. An alternative proof of

Erdős–Stone was given in 1975, & utilized Szemerédi regularity lemma, an essential technique in resolution of extremal graph theory problems.

4.2.2 Topics & concepts

- **Graph coloring.** Main article: Wikipedia/graph coloring. A *proper (vertex) coloring* of a graph G is a coloring of vertices of G s.t. no 2 adjacent vertices have the same color. Minimum number of colors needed to properly color G is called *chromatic number* of G, denoted $\chi(G)$. Determining chromatic number of specific graphs is a fundamental question in extremal graph theory, because many problems in area & related areas can be formulated in terms of graph coloring.

  2 simple lower bounds to chromatic number of a graph G is given by clique number $\omega(G)$ – all vertices of a clique must have distinct colors – & by $\frac{|V(G)|}{\alpha(G)}$, where $\alpha(G)$ is independence number, because set of vertices with a given color must form an independent set.

  A greedy coloring gives upper bound $\chi(G) \leq \Delta(G)+1$, where $\Delta(G)$ is maximum degree of G. When G is not an odd cycle or a clique, Brooks' theorem states: upper bound can be reduced to $\Delta(G)$. When G is a planar graph, 4-color theorem states: G has chromatic number ≤ 4.

  In general, determining whether a given graph has a coloring with a prescribed number of colors is known to be NP-hard.

  In addition to vertex coloring, other types of coloring are also studied, e.g. edge colorings. *Chromatic index* $\chi'(G)$ of a graph G is minimum number of colors in a proper edge-coloring of a graph, & Vizing's theorem states: chromatic index of a graph G is either $\Delta(G)$ or $\Delta(G)+1$.

- **Forbidden subgraphs.** Main article: Wikipedia/forbidden subgraph problem. *Forbidden subgraph problem* is 1 of central problems in extremal graph theory. Given a graph G, forbidden subgraph problem asks for maximal number of edges $\mathrm{ex}(n, G)$ in an n-vertex graph that does not contain a subgraph isomorphic to G.

  When $G = K_r$ is a complete graph, Turán's theorem gives an exact value for $\mathrm{ex}(n, K_r)$ & characterizes all graphs attaining this maximum; such graphs are known as Turán graphs. For non-bipartite graphs G, Erdős–Stone theorem gives an asymptotic value of $\mathrm{ex}(n, G)$ in terms of chromatic number of G. Problem of determining asymptotics of $\mathrm{ex}(n, G)$ when G is a bipartite graph is open; when G is a complete bipartite graph, this is known as Zarankiewicz problem.

- **Homomorphism density.** Main article: Wikipedia/Homomorphism density. *Homomorphism density* $t(H, G)$ of a graph H in a graph G describes probability that a randomly chosen map from vertex set of H to vertex set of G is also a graph homomorphism. It is closely related to *subgraph density*, which describes how often a graph H is found as a subgraph of G.

  Forbidden subgraph problem can be restated as maximizing edge density of a graph with G-density 0, & this naturally leads to generalization in form of *graph homomorphism inequalities*, which are inequalities relating $t(H, G)$ for various graphs H. By extending homomorphism density to graphons, which are objects that arise as a limit of dense graphs, graph homomorphism density can be written in form of integrals, & inequalities e.g. Cauchy–Schwarz inequality & Hölder's inequality can be used to derive homomorphism inequalities.

  A major open problem relating homomorphism densities is Sidorenko's conjecture, which states a tight lower bound on homomorphism density of a bipartite graph in a graph G in terms of edge density of G.

- **Graph regularity.** Main article: Wikipedia/Szemerédi regularity lemma. Edges between parts in a regular partition behave in a "random-like" fashion. *Szemerédi's regularity lemma* states: all graphs are 'regular' in sense: vertex set of any given graph can be partitioned into a bounded number of parts s.t. bipartite graph between most pairs of parts behave like random bipartite graphs. This partition gives a structural approximation to original graph, which reveals information about properties of original graph.

  Regularity lemma is a central result in extremal graph theory, & also has numerous applications in adjacent fields of additive combinatorics & commputational complexity theory. In addition to (Szemerédi) regularity, closely related notions of graph regularity e.g. strong regularity & Frieze-Kannan weak regularity have also been studied, as well as extensions of regularity to hypergraphs.

  Applications of graph regularity often utilize forms of counting lemmas & removal lemmas. In simplest forms, graph counting lemma uses regularity between pairs of parts in a regular partition to approximate number of subgraphs, & graph removal lemma states: given a graph with few copies of a given subgraph, can remove a small number of edges to eliminate all copies of subgraph." – Wikipedia/extremal graph theory

5 Miscellaneous

Tài liệu

[Gol18] Boris Goldengorin, ed. *Optimization problems in graph theory*. Vol. 139. Springer Optimization and Its Applications. In honor of Gregory Z. Gutin's 60th birthday. Springer, Cham, 2018, pp. xviii+331. ISBN: 978-3-319-94829-4; 978-3-319-94830-0. DOI: 10.1007/978-3-319-94830-0. URL: https://doi.org/10.1007/978-3-319-94830-0.

[Sha22] Shahriar Shahriari. *An invitation to combinatorics*. Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, 2022, pp. xv+613. ISBN: 978-1-108-47654-6.

[Val02] Gabriel Valiente. *Algorithms on trees and graphs*. Springer-Verlag, Berlin, 2002, pp. xiv+490. ISBN: 3-540-43550-6. DOI: 10.1007/978-3-662-04921-1. URL: https://doi.org/10.1007/978-3-662-04921-1.

[Val21] Gabriel Valiente. *Algorithms on trees and graphs—with Python code*. Texts in Computer Science. Second edition [of 1926815]. Springer, Cham, [2021] ©2021, pp. xv+386. ISBN: 978-3-303-81884-5; 978-3-303-81885-2. DOI: 10.1007/978-3-030-81885-2. URL: https://doi.org/10.1007/978-3-030-81885-2.