Combinatorial Optimization – Tối Ưu Tổ Hợp

Nguyễn Quản Bá Hồng*

Ngày 22 tháng 9 năm 2025

Tóm tắt nội dung

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

- *Combinatorial Optimization – Tối Ưu Tổ Hợp.*
  PDF: URL: .pdf.
  TEX: URL: .tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

1 Basic Combinatorial Optimization

1.1 [KV18]. Bernhard Korte, Jens Vygen. Combinatorial Optimization: Algorithms & Combinatorics. 6e

- Preface to 6e. Sect. 7.4 is devoted to shallow-light trees. Sect. 14.6 contains recent 2-factor approximation algorithm for submodule function maximization. Sect. 17.5 discusses Nemhauser-Ullmann algorithm & smoothed analysis. In Sect. 20.3, present $(\ln 4 + \epsilon)$-factor approximation algorithm for Steiner tree problem. Sect. 207 contains VPN theorem. There are also small additions, e.g., on integrality ratio in Sect. 5.1 & kernelization in Sect. 15.7.
- Preface to 5e. When preparing 1e of this book, >10 years ago, tried to accomplish 2 objectives: should be useful as an advanced graduate textbook but also as a reference work for research. With each new edition, have to decide how book can be improved further. Of course, less & less possible to describe growing area comprehensively.

  If included everything that we like, book would grow beyond a single volume. Since book is used for many courses, now even sometimes at undergraduate level, thought: adding some classical material might be more useful than including a selection of latest results.

*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.

In this edition, added a proof of Cayley's formula, more details on blocking flows, new faster b-matching separation algorithm, an approximation scheme for multidimensional knapsack, & results containing multicommodity max-flow min-cut ratio & sparest cut problem. There are further small improvements in numerous places & > 60 new exercises. Of course, also updated refs to point to most recent results & corrected some minor errors that were discovered.

- Preface to 4e. Again, have revised, updated, & significantly extended it for 4e. Have added some classical material that may have been missed so far, in particular on linear programming, network simplex algorithm, & max-cut problem. Have also added a number of new exercises & up-to-date references. Hope: these changes serve to make our book an even better basis for teaching & research. At http://www.or.uni-bonn.de/~vygen/co.html, continue to maintain updated information about this book.

- Preface to 3e. Most significant feature is a completely new chap on facility location. No constant-factor approximation algorithms were known for this important class of NP-hard problems until 8 years ago. Today there are several interesting & very different techniques that lead to good approximation guarantees, which makes this area particularly appealing also for teaching. In fact, chap has arisen from a special course on facility location.

  Many of other chaps have also been extended significantly. New material includes Fibonacci heaps, Fujishige's new maximum flow algorithm, flows over time, Schrijver's algorithm for submodular function minimization, & Robins-Zelikovsky Steiner tree approximation algorithm. Several proofs have been streamlined, & many new exercises & references have been added.

- Preface to 1e. Combinatorial optimization is 1 of youngest & most active areas of discrete mathematics & is probably its driving force today. It became a subject in its own right about 50 years ago. This book describes most important ideas, theoretical results, & algorithms in combinatorial optimization. Have conceived it as an advanced graduate text which can also be used as an up-to-date reference work for current research. Book includes essential fundamentals of graph theory, linear & integer optimization as well as very recent ones. Emphasis is on theoretical results & algorithms with provably good performance. Applications & heuristics are mentioned only occasionally.

  – Tối ưu hóa tổ hợp là 1 trong những lĩnh vực trẻ nhất & năng động nhất của toán học rời rạc & có lẽ là động lực thúc đẩy của nó ngày nay. Nó đã trở thành 1 môn học độc lập khoảng 50 năm trước. Cuốn sách này mô tả hầu hết các ý tưởng, kết quả lý thuyết, & thuật toán quan trọng nhất trong tối ưu hóa tổ hợp. Chúng tôi đã xây dựng nó như 1 giáo trình sau đại học nâng cao, đồng thời có thể được sử dụng làm tài liệu tham khảo cập nhật cho các nghiên cứu hiện tại. Sách bao gồm những kiến thức cơ bản thiết yếu về lý thuyết đồ thị, tối ưu hóa tuyến tính & số nguyên cũng như những kiến thức mới nhất. Trọng tâm là các kết quả lý thuyết & thuật toán có hiệu suất tốt đã được chứng minh. Các ứng dụng & phương pháp heuristic chỉ được đề cập đôi khi.

  Combinatorial optimization has its roots in combinatorics, operations research, & theoretical computer science. A main motivation: thousands of real-life problems can be formulated as abstract combinatorial optimization problems. Focus on detailed study of classical problems which occur in many different contexts, together with underlying theory.

  – Tối ưu hóa tổ hợp bắt nguồn từ toán học tổ hợp, nghiên cứu vận hành & khoa học máy tính lý thuyết. Động lực chính: hàng ngàn bài toán thực tế có thể được xây dựng thành các bài toán tối ưu hóa tổ hợp trừu tượng. Tập trung nghiên cứu chi tiết các bài toán cổ điển xảy ra trong nhiều bối cảnh khác nhau, cùng với lý thuyết nền tảng.

  Most combinatorial optimization problems can be formulated naturally in terms of graphs & as (integer) linear programs. Therefore this book starts, after an introduction, by reviewing basic graph theory & providing those results in linear & integer programming which are most relevant for combinatorial optimization.

  – Hầu hết các bài toán tối ưu hóa tổ hợp đều có thể được xây dựng 1 cách tự nhiên dưới dạng đồ thị & như các chương trình tuyến tính (số nguyên). Do đó, sau phần giới thiệu, cuốn sách này bắt đầu bằng việc xem xét lại lý thuyết đồ thị cơ bản & cung cấp những kết quả trong quy hoạch tuyến tính & số nguyên có liên quan nhất đến tối ưu hóa tổ hợp.

  Next classical topics in combinatorial optimization are studied: minimum spanning trees, shortest paths, network flows, matchings, & matroids. Most of problems discussed in Chaps. 6–14 have polynomial-time ("efficient") algorithms, while most of problems studied in Chaps. 15–21 are NP-hard, i.e., a polynomial-time algorithm is unlikely to exist. In many cases, one can at least find approximation algorithms that have a certain performance guarantee. Also mention some other strategies for coping with such "hard" problems.

  – Các chủ đề kinh điển tiếp theo trong tối ưu hóa tổ hợp sẽ được nghiên cứu: cây khung nhỏ nhất, đường đi ngắn nhất, luồng mạng, phép so khớp, & matroid. Hầu hết các bài toán được thảo luận trong Chương 6-14 đều có thuật toán thời gian đa thức ("hiệu quả"), trong khi hầu hết các bài toán được nghiên cứu trong Chương 15-21 đều là NP-khó, tức là, thuật toán thời gian đa thức khó có thể tồn tại. Trong nhiều trường hợp, ít nhất ta có thể tìm thấy các thuật toán xấp xỉ có 1 số đảm bảo hiệu suất nhất định. Ngoài ra, cũng đề cập đến 1 số chiến lược khác để giải quyết những bài toán "khó" như vậy.

  This book goes beyond scope of a normal textbook on combinatorial optimization in various aspects. E.g., cover equivalence of optimization & separation (for full-dimensional polytopes), $O(n^3)$-implementations of matching algorithms based on ear decompositions, Turing machines, perfect graph theorem, MAXSNP-hardness, Karmarkar-Karp algorithm for bin packing, recent approximation algorithms for multicommodity flows, survivable network design, & Euclidean traveling salesman problem. All results are accompanied by detailed proofs.

  – Cuốn sách này vượt ra ngoài phạm vi của 1 cuốn sách giáo khoa thông thường về tối ưu hóa tổ hợp ở nhiều khía cạnh. Ví dụ, bao gồm tính tương đương của tối ưu hóa & phân tách (cho đa diện toàn chiều), triển khai $O(n^3)$ của các thuật toán ghép dựa trên phân tích ear, máy Turing, định lý đồ thị hoàn hảo, độ khó MAXSNP, thuật toán Karmarkar-Karp cho đóng gói bin,

các thuật toán xấp xỉ gần đây cho luồng đa hàng hóa, thiết kế mạng sống sót, & Bài toán người bán hàng du lịch Euclid. Tất cả các kết quả đều kèm theo chứng minh chi tiết.

Of course, no book on combinatorial optimization can be absolutely comprehensive. Examples of topics mentioned only briefly or do not cover at all are tree decompositions, separators, submodular flows, path matchings, delta-matroids, matroid parity problem, location & scheduling problems, nonlinear problems, semidefinite programming, average-case analysis of algorithms, advanced data structures, parallel & randomized algorithms, & theory of probabilistically checkable proofs (cite PCP theorem without proof).

– Tất nhiên, không có cuốn sách nào về tối ưu hóa tổ hợp có thể hoàn toàn đầy đủ. Ví dụ về các chủ đề chỉ được đề cập ngắn gọn hoặc không đề cập đến bao gồm phân tích cây, bộ tách, luồng dưới mô-đun, khớp đường đi, delta-matroid, bài toán chẵn lẻ matroid, bài toán vị trí & lập lịch, bài toán phi tuyến tính, lập trình bán xác định, phân tích trường hợp trung bình của thuật toán, cấu trúc dữ liệu nâng cao, thuật toán song song & ngẫu nhiên, & lý thuyết về chứng minh kiểm tra xác suất (trích dẫn định lý PCP mà không cần chứng minh).

This book arose from several courses on combinatorial optimization & from special classes on topics like polyhedral combinatorics or approximation algorithms. Thus material for basic & advanced courses can be selected from this book.

– Cuốn sách này được hình thành từ 1 số khóa học về tối ưu hóa tổ hợp & từ các lớp học chuyên biệt về các chủ đề như tổ hợp đa diện hoặc thuật toán xấp xỉ. Do đó, tài liệu cho các khóa học cơ bản & nâng cao có thể được chọn từ cuốn sách này.

- 1. Introduction. Start with 2 examples. A company has a machine which drills holes into printed circuit boards. Since it produces many of these boards it wants machine to complete 1 board as fast as possible. Cannot optimize drilling time but can try to minimize time machine needs to move from 1 point to another. Usually drilling machines can move in 2 directions: table moves horizontally while drilling arm moves vertically. Since both movements can be done simultaneously, time needed to adjust machine from 1 position to another is proportional to maximum of horizontal & vertical distance. This is called l_∞-distance. (Older machines can only move either horizontally or vertically at a time; in this case adjusting time is proportional to l_1-distance, sum of horizontal & vertical distance.)

  – Bắt đầu với 2 ví dụ. Một công ty có 1 máy khoan lỗ trên bảng mạch in. Vì sản xuất nhiều bảng mạch in, công ty muốn máy hoàn thành 1 bảng mạch in càng nhanh càng tốt. Không thể tối ưu hóa thời gian khoan nhưng có thể cố gắng giảm thiểu thời gian máy cần di chuyển từ điểm này sang điểm khác. Thông thường, máy khoan có thể di chuyển theo 2 hướng: bàn máy di chuyển theo chiều ngang trong khi tay khoan di chuyển theo chiều dọc. Vì cả 2 chuyển động có thể được thực hiện đồng thời, thời gian cần thiết để điều chỉnh máy từ vị trí này sang vị trí khác tỷ lệ thuận với khoảng cách tối đa theo chiều ngang & chiều dọc. Khoảng cách này được gọi là l_∞-distance. (Các máy cũ hơn chỉ có thể di chuyển theo chiều ngang hoặc chiều dọc tại 1 thời điểm; trong trường hợp này, thời gian điều chỉnh tỷ lệ thuận với l_1-distance, tổng của khoảng cách theo chiều ngang & chiều dọc.)

  An optimum drilling path is given by an ordering of hole positions $p_1, \ldots, p_n$ s.t. $\sum_{i=1}^{n-1} d(p_i, p_{i+1})$ is minimum, where d is l_∞-distance: for 2 points $p = (x, y), p' = (x', y')$ in plane, write $d(p, p') = \max\{|x - x'|, |y - y'|\}$. An order of holes can be represented by a permutation, i.e., a bijection $\pi : [n] \to [n]$. Which permutation is best of course depends on hole positions; for each list of hole positions we have a different problem instance. Say: 1 instance of our problem is a list of points in plane, i.e., coordinates of holes to be drilled. Then problem can be stated formally as follows:

  – Đường khoan tối ưu được đưa ra bằng cách sắp xếp các vị trí lỗ $p_1, \ldots, p_n$ s.t. $\sum_{i=1}^{n-1} d(p_i, p_{i+1})$ là nhỏ nhất, trong đó d là khoảng cách l_∞: với 2 điểm $p = (x, y), p' = (x', y')$ trong mặt phẳng, hãy viết $d(p, p') = \max\{|x - x'|, |y - y'|\}$. Thứ tự các lỗ có thể được biểu diễn bằng 1 phép hoán vị, tức là 1 song ánh $\pi : [n] \to [n]$. Tất nhiên, phép hoán vị nào là tốt nhất phụ thuộc vào vị trí lỗ; với mỗi danh sách các vị trí lỗ, chúng ta có 1 trường hợp bài toán khác nhau. Giả sử: 1 trường hợp bài toán của chúng ta là 1 danh sách các điểm trong mặt phẳng, tức là tọa độ của các lỗ cần khoan. Khi đó, bài toán có thể được phát biểu 1 cách hình thức như sau:

  Problem 1 (Drilling problem)**.**

  Instance. *A set of points $p_1, \ldots, p_n \in \mathbb{R}^2$.*

  Task. *Find a permutation $\pi : [n] \to [n]$ s.t. $\sum_{i=1}^{n-1} d(p_{\pi(i)}, p_{\pi(i+1)})$ is minimum.*

  2nd example: Have a set of jobs to be done, each having a specified processing time. Each job can be done by a subset of employees, & assume: all employees who can do a job are equally efficient. Several employees can contribute to same job at same time, & 1 employee can contribute to several jobs (but not at same time). Objective: get all jobs done as early as possible.

  – VD2: Có 1 tập hợp các công việc cần thực hiện, mỗi công việc có thời gian xử lý cụ thể. Mỗi công việc có thể được thực hiện bởi 1 nhóm nhân viên, & giả định: tất cả nhân viên có thể thực hiện 1 công việc đều có hiệu suất như nhau. Nhiều nhân viên có thể cùng lúc thực hiện 1 công việc, & 1 nhân viên có thể thực hiện nhiều công việc (nhưng không cùng lúc). Mục tiêu: hoàn thành tất cả các công việc càng sớm càng tốt.

  In this model it suffices to prescribe for each employee how long he or she should work on which job. Order in which employees carry out their jobs is not important, since time when all jobs are done obviously depends only on maximum total working time we have assigned to 1 employee. Hence have to solve problem:

  – Trong mô hình này, chỉ cần quy định cho mỗi nhân viên thời gian làm việc cho từng công việc là đủ. Thứ tự nhân viên thực hiện công việc không quan trọng, vì thời gian hoàn thành tất cả công việc rõ ràng chỉ phụ thuộc vào tổng thời gian làm việc tối đa mà chúng ta đã phân bổ cho 1 nhân viên. Do đó, chúng ta phải giải quyết vấn đề:

Problem 2 (Job assignment problem)**.**

Instance. *A set of numbers $t_1, \dots, t_n \in \mathbb{R}_+$ (processing times for n jobs), a number $m \in \mathbb{N}$ of employees, & a nonempty subset $S_i \subset [m]$ of employees for each job $i \in [n]$.*

Task. *Find numbers $x_{ij} \in \mathbb{R}_+$, $\forall i \in [n], j \in S_i$ s.t. $\sum_{j \in S_i} x_{ij} = t_i$ for $i \in [n]$ & $\max_{j \in [m]} \sum_{i: j \in S_i} x_{ij}$ is minimum.*

These are 2 typical problems arising in combinatorial optimization. How to model a practical problem as an abstract combinatorial optimization problem is not described in this book; indeed there is no general recipe for this task. besides giving a precise formulation of input & desired output, often important to ignore irrelevant components (e.g., drilling time which cannot be optimized or order in which employees carry out their jobs).

– Đây là 2 bài toán điển hình phát sinh trong tối ưu hóa tổ hợp. Sách này không mô tả cách mô hình hóa 1 bài toán thực tế thành 1 bài toán tối ưu hóa tổ hợp trừu tượng; thực tế là không có công thức chung nào cho nhiệm vụ này. Bên cạnh việc đưa ra công thức chính xác về đầu vào & đầu ra mong muốn, việc bỏ qua các thành phần không liên quan (ví dụ: thời gian khoan không thể tối ưu hóa hoặc thứ tự thực hiện công việc của nhân viên) thường rất quan trọng.

Of course not interested in a solution to a particular drilling problem or job assignment problem in some company, but rather looking for a way how to solve all problems of these types. 1st consider Drilling prob.

– Tất nhiên là không quan tâm đến giải pháp cho 1 vấn đề khoan cụ thể hay vấn đề phân công công việc ở 1 công ty nào đó, mà là tìm cách giải quyết mọi vấn đề thuộc loại này. Trước tiên hãy xem xét vấn đề khoan.

- 1.1. Enumeration. How can a solution to Drilling problem look like? There are infinitely many instances (finite sets of points in plane), so cannot list an optimum permutation for each instance. Instead, what we look for is an algorithm which, given an instance, computes an optimum solution. Such an algorithm exists: Given a set of n points, just try all possible $n!$ orders, & for each compute l_∞-length of corresponding path.

  – Giải pháp cho bài toán Khoan có thể trông như thế nào? Có vô số trường hợp (tập hợp hữu hạn các điểm trên mặt phẳng), nên không thể liệt kê 1 hoán vị tối ưu cho từng trường hợp. Thay vào đó, chúng ta cần tìm 1 thuật toán, cho trước 1 trường hợp, tính toán 1 giải pháp tối ưu. Một thuật toán như vậy tồn tại: Cho 1 tập hợp n điểm, chỉ cần thử tất cả các bậc $n!$ có thể, & với mỗi trường hợp, hãy tính độ dài l_∞ của đường đi tương ứng.

  There are different ways of formulating an algorithm, differing mostly in level of detail & formal language they use. Certainly would not accept following as an algorithm: "Given a set of n points, find an optimum path & output it." Not specified at all how to find optimum solution. Above suggestion to enumerate all possible $n!$ orders is more useful, but still is not clear how to enumerate all orders. Here is 1 possible way: Enumerate all n-tuples of numbers $1, \dots, n$, i.e., all n^n vectors of $[n]^n$. This can be done similarly to counting: start with $(1, 1, \dots, 1), (1, \dots, 1, 2)$ up to $(1, \dots, 1, n)$ then switch to $(1, \dots, 1, 2, 1)$, & so on. At each step, increment last entry unless it is already n, in which case go back to last entry that is smaller than n, increment it & set all subsequent entries to 1. This techniques is sometimes called backtracking. Order in which vectors of $[n]^n$ are enumerated is called lexicographical order:

  – Có nhiều cách khác nhau để xây dựng 1 thuật toán, chủ yếu khác nhau ở mức độ chi tiết & ngôn ngữ hình thức mà họ sử dụng. Chắc chắn sẽ không chấp nhận sau đây là 1 thuật toán: "Cho 1 tập hợp n điểm, tìm 1 đường đi tối ưu & xuất ra nó." Không chỉ rõ cách tìm giải pháp tối ưu. Đề xuất ở trên để liệt kê tất cả các thứ tự $n!$ có thể hữu ích hơn, nhưng vẫn không rõ cách liệt kê tất cả các thứ tự. Sau đây là 1 cách có thể: Liệt kê tất cả các bộ n số $1, \dots, n$, tức là tất cả các vectơ n^n của $[n]^n$. Điều này có thể được thực hiện tương tự như đếm: bắt đầu với $(1, 1, \dots, 1), (1, \dots, 1, 2)$ đến $(1, \dots, 1, n)$ sau đó chuyển sang $(1, \dots, 1, 2, 1)$, & vân vân. Ở mỗi bước, tăng phần tử cuối cùng trừ khi nó đã là n, trong trường hợp đó, quay lại phần tử cuối cùng nhỏ hơn n, tăng phần tử đó & đặt tất cả các phần tử tiếp theo thành 1. Kỹ thuật này đôi khi được gọi là quay lui. Thứ tự liệt kê các vectơ của $[n]^n$ được gọi là thứ tự từ điển:

  Definition 1. *Let $x, y \in \mathbb{R}^n$: 2 vectors. Say: a vector x is* lexicographically smaller *than y if there exists an index $j \in [n]$ s.t. $x_i = y_i$ for $i \in [j-1]$ & $x_j < y_j$.*

  – Cho $x, y \in \mathbb{R}^n$: 2 vectơ. Giả sử: 1 vectơ x nhỏ hơn về mặt từ điển *so với y nếu tồn tại 1 chỉ số $j \in [n]$ s.t. $x_i = y_i$ với $i \in [j-1]$ & $x_j < y_j$.*

  Knowing how to enumerate all vectors of $[n]^n$, can simply check for each vector whether its entries are pairwise distinct &, if so, whether path represented by this vector is shorter than best path encountered so far.

  – Biết cách liệt kê tất cả các vectơ của $[n]^n$, có thể dễ dàng kiểm tra từng vectơ xem các mục của nó có phân biệt từng cặp hay không &, nếu có, liệu đường dẫn được biểu diễn bởi vectơ này có ngắn hơn đường dẫn tốt nhất đã gặp cho đến nay hay không.

  Since this algorithm enumerates n^n vectors it will take at least n^n steps (in fact, even more). This is not best possible. There are only $n!$ permutations of $[n]$, & $n!$ is significantly smaller than n^n. (By Stirling's formula $n! \approx \sqrt{2\pi n}\frac{n^n}{e^n}$ (Stirling [1730])). Shall show how to enumerate all paths in approximately $n^2 n!$ steps. Consider following algorithm which enumerates all permutations in lexicographical order:

  Problem 3 (Path enumeration algorithm)**.**

  Input. *A natural number $n \geq 3$. A set $\{p_1, \dots, p_n\}$ of points in plane.*

  Output. *A permutation $\pi^* : [n] \to [n]$ with $\text{cost}(\pi^*) \coloneqq \sum_{i=1}^{n-1} d(p_{\pi^*(i)}, p_{\pi^*(i+1)})$ minimum.*

p. 3+++

- 2. Graphs.
- 3. Linear Programming.
- 4. Linear Programming Algorithms.
- 5. Integer Programming.
- 6. Spanning Trees & Arborescences.
- 7. Shortest Paths.
- 8. Network Flows.
- 9. Minimum Cost Flows.
- 10. Maximum Matchings.
- 11. Weighted Matching.
- 12. b-Matchings & T-Joins.
- 13. Matroids.
- 14. Generalizations of Matroids.
- 15. NP-Completeness.
- 16. Approximation Algorithms.
- 17. Knapsack Problem.
- 18. Bin-Packing.
- 19. Multicommodity Flows & Edge-Disjoint Paths.
- 20. Network Design Problems.
- 21. Traveling Salesman Problem.
- 22. Facility Location.

2 Machine Learning for Combinatorial Optimization

2.1 Yoshua Bengio, Andrea Lodi, Antoine Prouvost. Machine Learning for Combinatorial Optimization: A Methodologiacl Tour d'Horizon. 2021

[2136 citations]

- Abstract. This paper surveys recent attempts, both from ML & operations research communities, at leveraging ML to solve combinatorial optimization problems. Given hard nature of these problems, state-of-art algorithms rely on handcrafted heuristics for making decisions that are otherwise too expensive to compute or mathematically not well defined. Thus, ML looks like a natural candidate to make such decisions in a more principled & optimized way. Advocate for pushing further integration of ML & combinatorial optimization & detail a methodology to do so. A main point of paper is seeing generic optimization problems as data points & inquiring what is relevant distribution of problems to use for learning on a given task.

  – Bài báo này khảo sát những nỗ lực gần đây, cả từ cộng đồng nghiên cứu ML & vận hành, trong việc tận dụng ML để giải quyết các bài toán tối ưu hóa tổ hợp. Do tính chất khó khăn của những bài toán này, các thuật toán tiên tiến dựa vào các phương pháp heuristic thủ công để đưa ra các quyết định mà nếu không sẽ quá tốn kém để tính toán hoặc chưa được định nghĩa rõ ràng về mặt toán học. Do đó, ML dường như là 1 ứng cử viên tự nhiên để đưa ra những quyết định như vậy theo cách có nguyên tắc & tối ưu hơn. Ủng hộ việc thúc đẩy tích hợp ML & tối ưu hóa tổ hợp & trình bày chi tiết phương pháp luận để thực hiện điều đó. Điểm chính của bài báo là xem các bài toán tối ưu hóa chung như các điểm dữ liệu & tìm hiểu xem phân phối các bài toán có liên quan nào để sử dụng cho việc học trên 1 nhiệm vụ nhất định.

- Keywords. Combinatorial optimization, ML, branch & bound, mixed-integer programming solvers.

- 1. Introduction. Operations research, also referred to as prescriptive analytics, started in 2nd world war as an initiative to use mathematics & computer science to assist military planners in their decisions (Fortun & Schweber, 1993). Nowadays, it is widely used in industry, including but not limited to transportation, supply chain, energy, finance, & scheduling. In this paper, focus on discrete optimization problems formulated as integer constrained optimization problems formulated as integer constrained optimization, i.e., with integral or binary variables (called *decision variables*). While not all such problems are hard to solve (e.g., shortest path problems), concentrate on combinatorial optimizations (NP-hard). This is bad news, in sense: for those problems, considered unlikely: an algorithm whose running time is polynomial in size of input exists. However, in practice, combinatorial optimization algorithms can solve instances with up to millions of decision variables & constraints.

  – Nghiên cứu hoạt động, còn được gọi là phân tích quy phạm, bắt đầu vào Thế chiến thứ 2 như 1 sáng kiến sử dụng toán học & khoa học máy tính để hỗ trợ các nhà hoạch định quân sự trong các quyết định của họ (Fortun & Schweber, 1993). Ngày nay, nó được sử dụng rộng rãi trong công nghiệp, bao gồm nhưng không giới hạn ở vận tải, chuỗi cung ứng, năng lượng, tài chính, & lập lịch. Trong bài báo này, tập trung vào các vấn đề tối ưu hóa rời rạc được xây dựng dưới dạng các vấn đề tối ưu hóa ràng buộc số nguyên được xây dựng dưới dạng tối ưu hóa ràng buộc số nguyên, tức là với các biến tích phân hoặc nhị phân (được gọi là *biến quyết định*). Mặc dù không phải tất cả các vấn đề như vậy đều khó giải (ví dụ: các vấn đề đường đi ngắn nhất), hãy tập trung vào tối ưu hóa tổ hợp (NP-khó). Theo 1 nghĩa nào đó, đây là tin xấu: đối với những vấn đề đó, được coi là không có khả năng xảy ra: tồn tại 1 thuật toán có thời gian chạy là đa thức về kích thước đầu vào. Tuy nhiên, trên thực tế, các thuật toán tối ưu hóa tổ hợp có thể giải quyết các trường hợp có tới hàng triệu biến quyết định & ràng buộc.

  How is it possible to solve NP-hard problems in practical time? Look at example of TSP, a NP-hard problem defined on a graph where we are searching for a cycle of minimum length visiting once & only once every node. A particular case is that of *Euclidean* TSP. In this version, each node is assigned coordinate in a plane [Or more generally in a vector space of arbitrary dimension.], & cost on an edge connecting 2 nodes is Euclidean distance between them. While theoretically as hard as general TSP, good approximate solution can be found more efficiently in Euclidean case by leveraging *structure* of graph (Larson & Odoni, 1981, Chapter 6.4.7). Likewise, diverse types of problems are solved by leveraging their special structure. Other algorithms, designed to be general, are found in hindsight to be empirically more efficient on particular problems types. Scientific literature covers rich set of techniques researchers have developed to tackle different combinatorial optimization problems. An expert will know how to further refine algorithm parameters to different behaviors of optimization process, thus extending this knowledge with unwritten intuition. These techniques, & parameters controlling them, have been collectively *learned* by community to perform on inaccessible distribution of problem instances deemed valuable. Focus of this paper is on combinatorial optimization algorithms that automatically perform learning on a chosen implicit distribution of problems. Incorporating ML components in algorithm can achieve this.

  – *Làm thế nào để giải quyết các bài toán NP-khó trong thời gian thực tế?* Hãy xem ví dụ về TSP, 1 bài toán NP-khó được định nghĩa trên đồ thị, trong đó chúng ta tìm kiếm 1 chu trình có độ dài nhỏ nhất, chỉ đi qua mỗi nút 1 lần. Một trường hợp cụ thể là TSP Euclid. Trong phiên bản này, mỗi nút được gán tọa độ trong 1 mặt phẳng [Hay tổng quát hơn là trong không gian vectơ có chiều tùy ý.], & chi phí trên 1 cạnh nối 2 nút là khoảng cách Euclid giữa chúng. Mặc dù về mặt lý thuyết cũng khó như TSP tổng quát, nhưng có thể tìm ra giải pháp gần đúng tốt hiệu quả hơn trong trường hợp Euclid bằng cách tận dụng *cấu trúc* của đồ thị (Larson & Odoni, 1981, Chương 6.4.7). Tương tự như vậy, nhiều loại bài toán khác nhau được giải quyết bằng cách tận dụng cấu trúc đặc biệt của chúng. Các thuật toán khác, được thiết kế để tổng quát, khi nhìn lại, được thấy là hiệu quả hơn về mặt thực nghiệm đối với các loại bài toán cụ thể. Tài liệu khoa học bao gồm nhiều kỹ thuật phong phú mà các nhà nghiên cứu đã phát triển để giải quyết các bài toán tối ưu hóa tổ hợp khác nhau. Một chuyên gia sẽ biết cách tinh chỉnh thêm các tham số thuật toán cho các hành vi khác nhau của quá trình tối ưu hóa, từ đó mở rộng kiến thức này bằng trực giác bất thành văn. Những kỹ thuật này, & các tham số điều khiển chúng, đã được cộng đồng *học tập* 1 cách tập thể để thực hiện trên phân phối không thể tiếp cận của các trường hợp bài toán được coi là có giá trị. Trọng tâm của bài báo này là các thuật toán tối ưu hóa tổ hợp tự động thực hiện học trên 1 phân phối ngầm định của các bài toán. Việc kết hợp các thành phần ML vào thuật toán có thể đạt được điều này.

  Conversely, ML focuses on performing a task given some (finite & usually noisy) data. It is well suited for natural signals for which no clear mathematical formulation emerges because true data distribution is not known analytically, e.g. when processing images, text, voice or molecules, or with recommender systems, social networks or financial predictions. Most of times, learning problem has a statistical formulation that is solved through mathematical optimization. Recently, dramatic progress has been achieved with DL, an ML subfield building large parametric approximators by composing simpler functions. DL excels when applied in high dimensional spaces with a large number of data points.

  – Ngược lại, Học máy (ML) tập trung vào việc thực hiện 1 tác vụ với 1 số dữ liệu nhất định (hữu hạn & thường bị nhiễu). Học máy rất phù hợp với các tín hiệu tự nhiên mà không có công thức toán học rõ ràng nào được đưa ra do phân phối dữ liệu thực sự không được biết rõ về mặt phân tích, ví dụ như khi xử lý hình ảnh, văn bản, giọng nói hoặc phân tử, hoặc với các hệ thống đề xuất, mạng xã hội hoặc dự đoán tài chính. Hầu hết các bài toán học máy đều có công thức thống kê được giải quyết thông qua tối ưu hóa toán học. Gần đây, đã đạt được những tiến bộ đáng kể với Học máy (DL), 1 lĩnh vực con của Học máy (ML) xây dựng các hàm xấp xỉ tham số lớn bằng cách kết hợp các hàm đơn giản hơn. DL tỏ ra vượt trội khi được áp dụng trong không gian đa chiều với số lượng điểm dữ liệu lớn.

  - 1.1. Motivation. From combinatorial optimization point of view, ML can help improve an algorithm on a distribution of problem instances in 2 ways. On 1 side, researcher assumes expert knowledge [Theoretical &/or empirical.] about optimization algorithm, but wants to replace some heavy computations by a fast approximation. Learning can be used to build such approximations in a generic way, i.e., without need to derive new explicit algorithms. On other side, expert knowledge may not be sufficient & some algorithmic decisions may be unsatisfactory. Goal: therefore to explore space of these decisions, & learn

out of this experience best performing behavior (policy), hopefully improving on state of art. Even though ML is approximate, will demonstrate through examples surveyed in this paper that this does not systematically mean: incorporating learning will compromise overall theoretical guarantees. From point of view of using ML to tackle a combinatorial problem, combinatorial optimization can decompose problem into smaller, hopefully simpler, learning tasks. Combinatorial optimization structure therefore acts as a relevant prior for model. It is also an opportunity to leverage combinatorial optimization literature, notably in terms of theoretical guarantees (e.g., feasibility & optimality).

– Từ góc độ tối ưu hóa tổ hợp, ML có thể giúp cải thiện thuật toán dựa trên phân phối các trường hợp bài toán theo 2 cách. Một mặt, nhà nghiên cứu giả định có kiến thức chuyên môn [lý thuyết &/hoặc thực nghiệm.] về thuật toán tối ưu hóa, nhưng muốn thay thế 1 số phép tính phức tạp bằng 1 phép xấp xỉ nhanh. Học máy có thể được sử dụng để xây dựng các phép xấp xỉ như vậy theo cách tổng quát, tức là không cần phải suy ra các thuật toán rõ ràng mới. Mặt khác, kiến thức chuyên môn có thể không đủ & 1 số quyết định thuật toán có thể không thỏa đáng. Mục tiêu: do đó, để khám phá không gian của những quyết định này, & rút ra từ kinh nghiệm này hành vi (chính sách) hiệu quả nhất, hy vọng sẽ cải thiện trình độ công nghệ. Mặc dù ML là gần đúng, nhưng sẽ chứng minh thông qua các ví dụ được khảo sát trong bài báo này rằng điều này không có nghĩa là 1 cách hệ thống: việc kết hợp học máy sẽ làm ảnh hưởng đến các đảm bảo lý thuyết tổng thể. Từ góc độ sử dụng ML để giải quyết 1 bài toán tổ hợp, tối ưu hóa tổ hợp có thể phân tích bài toán thành các nhiệm vụ học nhỏ hơn, hy vọng là đơn giản hơn. Do đó, cấu trúc tối ưu hóa tổ hợp đóng vai trò là 1 tiên nghiệm liên quan cho mô hình. Đây cũng là cơ hội để tận dụng tài liệu tối ưu hóa tổ hợp, đặc biệt là về mặt đảm bảo lý thuyết (ví dụ: tính khả thi & tính tối ưu).

- 1.2. Setting. Imagine a delivery company in Montreal that needs to solve TSP. Every day, customers may vary, but usually, many are downtown & few on top of Mont Royal mountain. Furthermore, Montreal streets are laid on a grid, making distances close to l_1 distance. How close? Not as much as Phoenix, but certainly more than Paris. Company does not care about solving all possible TSP, but only theirs. Explicitly defining what makes a TSP a likely one for company is tedious, does not scale, & not clear hot it can be leveraged when explicitly writing an optimization algorithm. Would like to automatically specialize TSP algorithms for this company.

  – Hãy tưởng tượng 1 công ty giao hàng ở Montreal cần giải quyết bài toán TSP. Mỗi ngày, khách hàng có thể khác nhau, nhưng thường thì phần lớn tập trung ở trung tâm thành phố & 1 số ít trên đỉnh núi Mont Royal. Hơn nữa, đường phố Montreal được bố trí theo dạng lưới, khiến khoảng cách gần bằng 1_1. Gần đến mức nào? Không gần bằng Phoenix, nhưng chắc chắn gần hơn Paris. Công ty không quan tâm đến việc giải quyết tất cả các bài toán TSP khả thi, mà chỉ quan tâm đến bài toán của họ. Việc xác định rõ ràng điều gì khiến 1 bài toán TSP trở nên khả thi đối với công ty là rất phức tạp, không thể mở rộng quy mô, & không rõ ràng về khả năng tận dụng nó khi viết 1 thuật toán tối ưu hóa cụ thể. Tôi muốn tự động chuyên môn hóa các thuật toán TSP cho công ty này.

  True probability distribution of likely TSP in Montreal scenario is defining instances on which we would like our algorithm to perform well. This is unknown, & cannot even be mathematically characterized in an explicit way. Because we do not know what is in this distribution, we can only learn an algorithm that performs well on a finite set of TSP sampled from this distribution (e.g., a set of historical instances collected by company), thus implicitly incorporating desired information abut distribution of instances. As a comparison, in traditional ML tasks, true distribution could be that of all possible images of cats, while training distribution is a finite set of such images. Challenge in learning: an algorithm that performs well on problem instances used for learning may not work properly on other instances from true probability distribution. For company, this would mean algorithm only does well on past problems, but not on future ones. To control this, monitor performance of learned algorithm over another independent set of *unseen* problem instances. Keeping performances similar between instances used for learning & unseen ones is known in ML as *generalizing*. Current ML algorithms can generalize to examples from same distribution, but tend to have more difficulty generalizing out-of-distribution (although this is a topic of intense research in ML), & so we may expect combinatorial optimization algorithms that leverage ML models to fail when evaluated on unseen problem instances that are too far from what has been used for training ML predictor. As previously motivated, also worth noting: traditional combinatorial optimization algorithms might not even work consistently across all possible instances of a problem family, but rather tend to be more adapted to particular structures of problems, e.g., Euclidean TSP.

  – Phân phối xác suất thực của TSP khả dĩ trong kịch bản Montreal đang xác định các trường hợp mà chúng ta muốn thuật toán của mình hoạt động tốt. Điều này là chưa biết, & thậm chí không thể được mô tả toán học 1 cách rõ ràng. Vì chúng ta không biết những gì nằm trong phân phối này, chúng ta chỉ có thể học 1 thuật toán hoạt động tốt trên 1 tập hợp hữu hạn các TSP được lấy mẫu từ phân phối này (ví dụ: 1 tập hợp các trường hợp lịch sử do công ty thu thập), do đó ngầm kết hợp thông tin mong muốn về phân phối của các trường hợp. Để so sánh, trong các tác vụ ML truyền thống, phân phối thực có thể là phân phối của tất cả các hình ảnh mèo có thể có, trong khi phân phối huấn luyện là 1 tập hợp hữu hạn các hình ảnh như vậy. Thách thức trong học tập: 1 thuật toán hoạt động tốt trên các trường hợp bài toán được sử dụng để học có thể không hoạt động chính xác trên các trường hợp khác theo phân phối xác suất thực. Đối với công ty, điều này có nghĩa là thuật toán chỉ hoạt động tốt trên các bài toán trong quá khứ, nhưng không hoạt động tốt trên các bài toán trong tương lai. Để kiểm soát điều này, hãy theo dõi hiệu suất của thuật toán đã học trên 1 tập hợp độc lập khác gồm các trường hợp bài toán *unseen*. Việc duy trì hiệu suất tương tự giữa các trường hợp được sử dụng để học & unseen trong ML được gọi là *generalizing*. Các thuật toán ML hiện tại có thể khái quát hóa các ví dụ từ cùng 1 phân phối, nhưng thường gặp khó khăn hơn trong việc khái quát hóa các ví dụ ngoài phân phối (mặc dù đây là 1 chủ đề nghiên cứu chuyên sâu trong ML), & vì vậy chúng ta có thể dự đoán các thuật toán tối ưu hóa tổ hợp tận dụng các mô hình ML sẽ thất bại khi được đánh giá trên các trường hợp bài toán chưa từng thấy, quá khác biệt so với những gì đã được sử dụng để huấn luyện bộ dự đoán ML. Như đã đề cập trước đó, cũng đáng lưu ý: các thuật toán tối ưu hóa tổ hợp truyền thống thậm chí có thể không hoạt

động nhất quán trên tất cả các trường hợp có thể có của 1 họ bài toán, mà thay vào đó có xu hướng thích ứng hơn với các cấu trúc bài toán cụ thể, ví dụ: Euclidean TSP.

Finally, implicit knowledge extracted by ML algorithms is complementary to hard-won explicit expertise extracted through combinatorial optimization research. Rather, it aims to augment & automate unwritten expert intuition (or lack of) on various existing algorithms. Given: these problems are highly structured, believe: relevant to augment solving algorithms with ML – & especially DP to address high dimensionality of such problems.

– Cuối cùng, tri thức ngầm được trích xuất bởi các thuật toán ML bổ sung cho kiến thức chuyên môn rõ ràng khó khăn thu được thông qua nghiên cứu tối ưu hóa tổ hợp. Thay vào đó, nó hướng đến việc tăng cường & tự động hóa trực giác chuyên gia chưa được viết ra (hoặc thiếu) trên nhiều thuật toán hiện có. Do các bài toán này có cấu trúc cao, nên tin rằng: có liên quan đến việc tăng cường giải quyết các thuật toán bằng ML – & đặc biệt là DP để giải quyết các bài toán có tính đa chiều cao như vậy.

In following, survey attempts in literature to achieve such automation & augmentation, & present a methodological overview of those approaches. In light of current state of field, literature we survey is exploratory, i.e., aim at highlighting promising research directions in use of ML within combinatorial optimization, instead of reporting on already mature algorithms.

– Sau đây, chúng tôi khảo sát các tài liệu nhằm đạt được sự tự động hóa & tăng cường này, đồng thời trình bày tổng quan về phương pháp luận của các phương pháp đó. Xét đến tình hình hiện tại của lĩnh vực này, tài liệu chúng tôi khảo sát mang tính chất thăm dò, tức là nhằm mục đích làm nổi bật các hướng nghiên cứu đầy hứa hẹn trong việc sử dụng ML trong tối ưu hóa tổ hợp, thay vì báo cáo về các thuật toán đã hoàn thiện.

- 1.3. Outline. Have introduced context & motivations for building combinatorial optimization algorithms together with ML. Remainder of this paper is organized as follows. Sect. 2 provides minimal prerequisites in combinatorial optimization, ML, DL, & RL necessary to fully grasp content of paper. Sect. 3 surveys recent literature & derives 2 distinctive, orthogonal, views: Sect. 3.1 shows how ML policies can either be learned by imitating an expert or discovered through experience, while Sect. 3.2 discuses interplay between ML & combinatorial optimization components. Sect. 5 pushes further reflection on use of ML for combinatorial optimization & brings to fore some methodological points. In Sect. 6, detail critical practical challenges of field. Finally, some conclusions are drawn in Sect. 7.

  – Đã giới thiệu bối cảnh & động lực để xây dựng các thuật toán tối ưu hóa tổ hợp cùng với ML. Phần còn lại của bài báo này được tổ chức như sau. Mục 2 cung cấp các điều kiện tiên quyết tối thiểu trong tối ưu hóa tổ hợp, ML, DL, & RL cần thiết để nắm bắt đầy đủ nội dung của bài báo. Mục 3 khảo sát các tài liệu gần đây & đưa ra 2 quan điểm riêng biệt, trực giao: Mục 3.1 cho thấy cách các chính sách ML có thể được học bằng cách bắt chước 1 chuyên gia hoặc được khám phá thông qua kinh nghiệm, trong khi Mục 3.2 thảo luận về sự tương tác giữa các thành phần ML & tối ưu hóa tổ hợp. Mục 5 thúc đẩy sự suy ngẫm sâu hơn về việc sử dụng ML cho tối ưu hóa tổ hợp & làm nổi bật 1 số điểm về phương pháp luận. Trong Mục 6, chi tiết những thách thức thực tế quan trọng của lĩnh vực này. Cuối cùng, 1 số kết luận được rút ra trong Mục 7.

- 2. Preliminaries. In this sect, give a basic (sometimes rough) overview of combinatorial optimization & ML, with unique aim of introducing concepts that are strictly required to understand remainder of paper.

  – Trong phần này, cung cấp tổng quan cơ bản (đôi khi sơ lược) về tối ưu hóa tổ hợp & ML, với mục đích duy nhất là giới thiệu các khái niệm bắt buộc để hiểu phần còn lại của bài báo.

  - 2.1. Combinatorial optimization. W.l.o.g., a combinatorial optimization problem can be formulated as a constrained min-operation program. Constraints model natural or imposed restrictions of problem, variables define decisions to be made, while objective function, generally a cost to be minimized, defines measure of quality of every feasible assignment of values to variables. If objective & constraints are linear, problem is called a linear programming problem. If, in addition, some variables are also restricted to only assume integer values, then problem is a mixed-integer linear programming problem.

    – Ví dụ, 1 bài toán tối ưu hóa tổ hợp có thể được xây dựng dưới dạng 1 chương trình min-operation ràng buộc. Các ràng buộc mô hình hóa các hạn chế tự nhiên hoặc áp đặt của bài toán, các biến xác định các quyết định cần đưa ra, trong khi hàm mục tiêu, thường là chi phí cần tối thiểu hóa, xác định thước đo chất lượng của mọi phép gán giá trị khả thi cho các biến. Nếu ràng buộc mục tiêu & là tuyến tính, bài toán được gọi là bài toán quy hoạch tuyến tính. Nếu, ngoài ra, 1 số biến cũng bị giới hạn chỉ nhận các giá trị nguyên, thì bài toán được gọi là bài toán quy hoạch tuyến tính hỗn hợp số nguyên.

    Set of points that satisfy constraints i feasible region. Every point in that set (often referred to as a feasible solution) yields an upper bound on objective value of optimal solution. Exact solving is an important aspect of field, hence a lot of attention is also given to find lower bounds to optimal cost. Tighter lower bounds, w.r.t. optimal solution value, higher chances that current algorithmic approaches to tackle mixed-integer linear programming described in following could be successful, i.e., effective if not efficient.

    – Tập hợp các điểm thỏa mãn các ràng buộc trong miền khả thi. Mỗi điểm trong tập hợp đó (thường được gọi là 1 nghiệm khả thi) tạo ra 1 giới hạn trên của giá trị mục tiêu của nghiệm tối ưu. Giải chính xác là 1 khía cạnh quan trọng của lĩnh vực này, do đó việc tìm kiếm giới hạn dưới của chi phí tối ưu cũng được chú trọng. Giới hạn dưới càng chặt chẽ, liên quan đến giá trị nghiệm tối ưu, thì khả năng thành công của các phương pháp tiếp cận thuật toán hiện tại để giải quyết bài toán quy hoạch tuyến tính hỗn hợp được mô tả dưới đây càng cao, tức là hiệu quả nếu không muốn nói là hiệu suất.

    Linear & mixed-integer linear programming problems are workhorse of combinatorial optimization because they can model a wide variety of problems & are best understood, i.e., there are reliable algorithms & software tools to solve them. Give them special considerations in this paper but, of course, they do not represent entire combinatorial optimization, mixed-integer

nonlinear programming being a rapidly expanding & very significant area both in theory & in practical applications. W.r.t. complexity & solution methods, linear programming is a polynomial problem, well solved, in theory & in practice, through simplex algorithm or interior points methods. Mixed-integer linear programming, on other hand, is an NP-hard problem,, which does not make it hopeless. Indeed, easy to see: complexity of mixed-integer linear programming is associated with integrality requirement on (some of) variables, which makes mixed-integer linear programming feasible region nonconvex. However, dropping integrality requirement (i) defines a proper relaxation of mixed-integer linear programming (i.e., an optimization problem whose feasible region contains mixed-integer linear programming feasible region), which (ii) happens to be an linear programming, i.e., polynomially solvable. This immediately suggests algorithmic line of attach that is used to solve mixed-integer linear programming through a whole ecosystem of branch-&-bound techniques to perform implicit enumeration. Branch & bound implements a divide-&-conquer type of algorithm representable by a search tree in which, at every node, a linear programming relaxation of problem (possibly augmented by branching decisions, see below) is efficiently computed. If relaxation is infeasible, or if solution of relaxation is naturally (mixed-)integer, i.e., mixed-integer linear programming feasible, node does not need to be expanded. Otherwise, there exists at least 1 variable, among those supposed to be integer, taking a fractional value in linear programming solution & that variable can be chosen for branching (enumeration), i.e., by restricting its value in such a way that 2 child nodes are created. 2 child nodes have disjoint feasible regions, none of which contains solution of previous linear programming relaxation. Use Fig. 1: A branch-&-bound tree for mixed-integer linear programming. Linear programming relaxation is computed at every node (only partially shown in figure). Nodes still open for exploration are represented as blank. to illustrate branch-&-bound algorithm for a minimization mixed-integer linear programming. At root node in figure, variable x_2 has a fractional value in linear programming solution (not represented), thus branching is done on floor (here 0) & ceiling (here 1) of this value. When an integer solution is found, also get an upper bound, denoted as $\overline{z}$, on optimal solution value of problem. At every node, can then compare solution value of relaxation, denoted as $\underline{z}$, with minimum upper bound found so far, called *incumbent solution value*. If latter is smaller than former for a specific node, no better (mixed-)integer solution can be found in sub-tree originated by node itself, & it can be pruned.

– Các bài toán quy hoạch tuyến tính số nguyên hỗn hợp là công cụ đắc lực của tối ưu hóa tổ hợp vì chúng có thể mô hình hóa nhiều loại bài toán khác nhau & được hiểu rõ nhất, tức là có các thuật toán & công cụ phần mềm đáng tin cậy để giải quyết chúng. Hãy cân nhắc đặc biệt trong bài báo này nhưng tất nhiên, chúng không đại diện cho toàn bộ tối ưu hóa tổ hợp, quy hoạch phi tuyến số nguyên hỗn hợp là 1 lĩnh vực đang phát triển nhanh chóng & rất quan trọng cả về lý thuyết & ứng dụng thực tế. Về độ phức tạp & phương pháp giải, quy hoạch tuyến tính là 1 bài toán đa thức, được giải tốt, về lý thuyết & thực hành, thông qua thuật toán đơn hình hoặc phương pháp điểm trong. Mặt khác, quy hoạch tuyến tính số nguyên hỗn hợp là 1 bài toán NP-khó, điều này không làm cho nó trở nên vô vọng. Thật vậy, dễ thấy: độ phức tạp của quy hoạch tuyến tính số nguyên hỗn hợp có liên quan đến yêu cầu tính toàn vẹn trên (một số) biến, điều này làm cho quy hoạch tuyến tính số nguyên hỗn hợp khả thi trong vùng không lồi. Tuy nhiên, việc loại bỏ yêu cầu tích phân (i) xác định 1 sự nới lỏng thích hợp của quy hoạch tuyến tính hỗn hợp (tức là 1 bài toán tối ưu hóa mà miền khả thi của nó chứa miền khả thi quy hoạch tuyến tính hỗn hợp), mà (ii) tình cờ là 1 quy hoạch tuyến tính, tức là có thể giải được bằng đa thức. Điều này ngay lập tức gợi ý dòng thuật toán đính kèm được sử dụng để giải quyết quy hoạch tuyến tính hỗn hợp thông qua toàn bộ hệ sinh thái các kỹ thuật ràng buộc nhánh để thực hiện liệt kê ngầm định. Ràng buộc nhánh triển khai 1 loại thuật toán chia để trị có thể biểu diễn bằng 1 cây tìm kiếm trong đó, tại mỗi nút, 1 sự nới lỏng quy hoạch tuyến tính của bài toán (có thể được tăng cường bằng các quyết định phân nhánh, xem bên dưới) được tính toán hiệu quả. Nếu sự nới lỏng là không khả thi, hoặc nếu giải pháp của sự nới lỏng là số nguyên (hỗn hợp) tự nhiên, tức là khả thi quy hoạch tuyến tính hỗn hợp, thì nút không cần phải được mở rộng. Nếu không, tồn tại ít nhất 1 biến, trong số các biến được cho là số nguyên, lấy giá trị phân số trong giải pháp quy hoạch tuyến tính & biến đó có thể được chọn để phân nhánh (liệt kê), tức là bằng cách hạn chế giá trị của nó theo cách tạo ra 2 nút con. 2 nút con có các vùng khả thi không giao nhau, không có vùng nào chứa giải pháp của phép giãn quy hoạch tuyến tính trước đó. Sử dụng Hình 1: Cây có giới hạn nhánh cho quy hoạch tuyến tính số nguyên hỗn hợp. Độ giãn quy hoạch tuyến tính được tính tại mọi nút (chỉ hiển thị 1 phần trong hình). Các nút vẫn mở để khám phá được biểu diễn là trống. để minh họa thuật toán có giới hạn nhánh cho quy hoạch tuyến tính số nguyên hỗn hợp tối thiểu hóa. Tại nút gốc trong hình, biến x_2 có giá trị phân số trong giải pháp quy hoạch tuyến tính (không được biểu diễn), do đó việc phân nhánh được thực hiện trên sàn (ở đây là 0) & trần (ở đây là 1) của giá trị này. Khi tìm thấy 1 giải pháp số nguyên, cũng lấy 1 giới hạn trên, được ký hiệu là $\overline{z}$, trên giá trị giải pháp tối ưu của bài toán. Tại mỗi nút, có thể so sánh giá trị nghiệm thư giãn, ký hiệu là $\underline{z}$, với cận trên nhỏ nhất tìm được cho đến nay, được gọi là *giá trị nghiệm hiện tại*. Nếu nghiệm sau nhỏ hơn nghiệm trước đối với 1 nút cụ thể, không thể tìm thấy nghiệm nguyên (hỗn hợp) nào tốt hơn trong cây con do chính nút đó tạo ra, & nó có thể được cắt tỉa.

All commercial & noncommercial mixed-integer linear programming solvers enhance above enumeration framework with extensive use of cutting planes, i.e., valid linear inequalities that are added to original formulation (especially at root of branch-&-bound tree) in attempt of strengthening its linear programming relaxation. Resulting framework, referred to as branch-&-cut algorithm, is then further enhanced by additional algorithmic components, preprocessing & primal heuristics being most crucial ones. Reader is referred to Wolsey (1998) & Conforti, Conrnuéjols, & Zambelli (2014) for extensive textbooks on mixed-integer linear programming & to Lodi (2009) for a detailed description of algorithmic components of mixed-integer linear programming solvers.

– Tất cả các bộ giải quy hoạch tuyến tính hỗn hợp số nguyên thương mại & phi thương mại đều cải tiến khuôn khổ liệt kê trên bằng cách sử dụng rộng rãi các mặt phẳng cắt, tức là các bất đẳng thức tuyến tính hợp lệ được thêm vào công thức ban đầu (đặc biệt là tại gốc của cây nhánh-cắt) nhằm mục đích củng cố tính chất nới lỏng quy hoạch tuyến tính của nó. Khuôn khổ kết quả, được gọi là thuật toán cắt nhánh, sau đó được cải tiến hơn nữa bằng các thành phần thuật toán bổ

sung, trong đó tiền xử lý & các phương pháp heuristic nguyên thủy là quan trọng nhất. Độc giả có thể tham khảo Wolsey (1998) & Conforti, Conrnuéjols & Zambelli (2014) để biết thêm các sách giáo khoa phong phú về quy hoạch tuyến tính hỗn hợp & Lodi (2009) để biết thêm mô tả chi tiết về các thành phần thuật toán của bộ giải quy hoạch tuyến tính hỗn hợp số nguyên.

End sect by noting: there is a vast literature devoted to (primal) heuristics, i.e., algorithms designed to compute "good in practice" solutions to CO problems without optimality guarantee. Although a general discussion on them is outside scope here, those heuristic methods play a central role in combinatorial optimization & will be considered in specific contexts in present paper. Interested reader is referred to Fischetti & Lodi (2011) & Gendreau & Potvin (2010).

– Kết thúc phần này bằng cách lưu ý: có rất nhiều tài liệu chuyên sâu về phương pháp heuristic (nguyên thủy), tức là các thuật toán được thiết kế để tính toán các giải pháp "tốt trong thực tế" cho các bài toán CO mà không cần đảm bảo tính tối ưu. Mặc dù thảo luận chung về chúng nằm ngoài phạm vi bài viết này, nhưng các phương pháp heuristic này đóng vai trò trung tâm trong tối ưu hóa tổ hợp & sẽ được xem xét trong các bối cảnh cụ thể của bài báo này. Độc giả quan tâm có thể tham khảo Fischetti & Lodi (2011) & Gendreau & Potvin (2010).

- 2.2. ML.

  * Supervised learning. In supervised learning, a set of input (features)/target pairs is provided & task: find a function that every input has a predicted output as close as possible to provided target. Finding such a function is called learning & is solved through an optimization problem over a family of functions. Loss function, i.e., measure of discrepancy between output & target, can be chosen depending on task (regression, classification, etc.) & on optimization methods. However, this approach is not enough because problem has a statistical nature. It is usually easy enough to achieve a good score on given examples but one wants to achieve a good score on unseen examples (test data). This is known as generalization.
  – Trong học có giám sát, 1 tập hợp các cặp đầu vào (đặc trưng)/mục tiêu được cung cấp & nhiệm vụ: tìm 1 hàm mà mọi đầu vào đều có đầu ra dự đoán càng gần với mục tiêu đã cho càng tốt. Việc tìm ra 1 hàm như vậy được gọi là học & được giải quyết thông qua 1 bài toán tối ưu hóa trên 1 họ các hàm. Hàm mất mát, tức là thước đo sự khác biệt giữa đầu ra & mục tiêu, có thể được lựa chọn tùy thuộc vào nhiệm vụ (hồi quy, phân loại, v.v.) & phương pháp tối ưu hóa. Tuy nhiên, cách tiếp cận này là chưa đủ vì bài toán có bản chất thống kê. Thông thường, việc đạt điểm cao trong các ví dụ cho sẵn khá dễ dàng, nhưng người ta muốn đạt điểm cao trong các ví dụ chưa được biết đến (dữ liệu kiểm tra). Điều này được gọi là khái quát hóa.
  Mathematically speaking, let X, Y, following a joint probability distribution P, be random variables representing input features & target. Let l be per sample loss function to minimize, & let $\{f_\theta; \theta \in \mathbb{R}^p\}$ be family of ML models (parametric in this case) to optimize over. Supervised learning problem is framed as (1)

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{X,Y \sim P} l(Y, f_\theta(X)).$$

  E.g., f_θ could be a linear model with weights θ that we wish to *learn*. Loss function f is task dependent (e.g., classification error) & can sometimes be replaced by a surrogate one (e.g., a differentiable one). Probability distribution is unknown & inaccessible. E.g., it can be probability distribution of all natural images. Therefore, it is approximated by empirical probability distribution over a finite dataset $D_{\text{train}} = \{(x_i, y_i)\}_i$ & optimization problem solved is (2)

$$\min_{\theta \in \mathbb{R}^p} \sum_{(x,y) \in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} l(y, f_\theta(x)).$$

  A model is said to generalize, if low objective values of (2) translate in low objective values of (1). Because (1) remains inaccessible, estimate generalization error by evaluating trained model on a separate test dataset D_{test} with (3)

$$\sum_{(x,y) \in D_{\text{test}}} \frac{1}{|D_{\text{test}}|} l(y, f_\theta(x)).$$

  – Về mặt toán học, cho X, Y, tuân theo phân phối xác suất kết hợp P, là các biến ngẫu nhiên biểu diễn các đặc trưng đầu vào & mục tiêu. Cho l là hàm mất mát trên mỗi mẫu để tối thiểu hóa, & cho $\{f_\theta; \theta \in \mathbb{R}^p\}$ là họ các mô hình ML (tham số trong trường hợp này) để tối ưu hóa. Bài toán học có giám sát được đóng khung như sau (1)

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}_{X,Y \sim P} l(Y, f_\theta(X)).$$

  Ví dụ: f_θ có thể là 1 mô hình tuyến tính với các trọng số θ mà chúng ta muốn *học*. Hàm mất mát f phụ thuộc vào tác vụ (ví dụ: lỗi phân loại) & đôi khi có thể được thay thế bằng 1 hàm thay thế (ví dụ: hàm khả vi). Phân phối xác suất không xác định & không thể truy cập được. Ví dụ: nó có thể là phân phối xác suất của tất cả các ảnh tự nhiên. Do đó, nó được xấp xỉ bằng phân phối xác suất thực nghiệm trên 1 tập dữ liệu hữu hạn $D_{\text{train}} = \{(x_i, y_i)\}_i$ & bài toán tối ưu hóa được giải quyết là (2)

$$\min_{\theta \in \mathbb{R}^p} \sum_{(x,y) \in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} l(y, f_\theta(x)).$$

  Một mô hình được gọi là tổng quát hóa nếu các giá trị mục tiêu thấp của (2) chuyển thành các giá trị mục tiêu thấp của (1). Vì (1) vẫn không thể truy cập được, hãy ước tính lỗi tổng quát bằng cách đánh giá mô hình đã được huấn luyện trên

1 tập dữ liệu thử nghiệm riêng biệt D_{test} với (3)

$$\sum_{(x,y)\in D_{\text{test}}} \frac{1}{|D_{\text{test}}|} l(y, f_\theta(x)).$$

If a model (i.e., a family of functions) can represent many different functions, model is said to have high capacity & is prone to overfitting: doing well on training data but not generalizing to test data. Regularization is anything that can improve test score at expense of training score & is used to restrict practical capacity of a model. On contrary, if capacity is too low, model underfits & performs poorly on both sets. Boundary between overfitting & underfitting can be estimated by changing effective capacity (richness of family of functions reachable by training): below critical capacity one underfits & test error decreases with increases in capacity, while above that critical capacity one overfits & test error increases with increases in capacity.

– Nếu 1 mô hình (tức là 1 họ hàm) có thể biểu diễn nhiều hàm khác nhau, mô hình được cho là có khả năng cao & dễ bị quá khớp: hoạt động tốt trên dữ liệu huấn luyện nhưng không tổng quát hóa được trên dữ liệu kiểm tra. Chính quy hóa là bất kỳ điều gì có thể cải thiện điểm kiểm tra bằng cách đánh đổi điểm huấn luyện & được sử dụng để hạn chế khả năng thực tế của 1 mô hình. Ngược lại, nếu khả năng quá thấp, mô hình sẽ không khớp & hoạt động kém trên cả 2 tập. Ranh giới giữa quá khớp & không khớp có thể được ước tính bằng cách thay đổi khả năng hiệu dụng (độ phong phú của họ hàm có thể đạt được bằng huấn luyện): dưới khả năng tới hạn, 1 mô hình sẽ không khớp & lỗi kiểm tra giảm khi khả năng tăng, trong khi trên khả năng tới hạn, 1 mô hình sẽ không khớp & lỗi kiểm tra tăng khi khả năng tăng.

Selecting the best among various trained models cannot be done on test set. Selection is a form of optimization, & doing so on test set would bias estimator in (2). This is a common form of data dredging, & a mistake to be avoided. To perform model selection, a validation dataset D_{valid} is used to estimate generalization error of different ML models is necessary. Model selection can be done based on these estimates, & final unbiased generalization error of selected model can be computed on test set. Validation set is therefore often used to select effective capacity, e.g., by changing amount of training, number of parameters θ, & amount of regularization imposed to model.

– Việc lựa chọn mô hình tốt nhất trong số các mô hình đã được huấn luyện khác nhau không thể thực hiện trên tập kiểm tra. Lựa chọn là 1 dạng tối ưu hóa, & việc thực hiện như vậy trên tập kiểm tra sẽ làm sai lệch ước lượng trong (2). Đây là 1 dạng khai thác dữ liệu phổ biến, & 1 lỗi cần tránh. Để thực hiện lựa chọn mô hình, cần sử dụng 1 tập dữ liệu xác thực D_{valid} để ước tính lỗi tổng quát hóa của các mô hình ML khác nhau. Lựa chọn mô hình có thể được thực hiện dựa trên các ước tính này, & lỗi tổng quát hóa không thiên vị cuối cùng của mô hình đã chọn có thể được tính toán trên tập kiểm tra. Do đó, tập xác thực thường được sử dụng để lựa chọn năng lực hiệu quả, ví dụ: bằng cách thay đổi lượng dữ liệu huấn luyện, số lượng tham số θ, & lượng chính quy hóa áp dụng cho mô hình.

∗ **Unsupervised learning.** In unsupervised learning, one does not have targets for task one wants to solve, but rather tries to capture some characteristics of joint distribution of observed random variables. Variety of tasks include density estimation, dimensionality reduction, & clustering. Because unsupervised learning has received so far little attention in conjunction with combinatorial optimization & its immediate use seems difficult, we are not discussing it any further. Reader is referred to Bishop (2006); Goodfellow, Bengio, & Courville (2016); Murphy (2012) for textbooks on ML.

– **Học không giám sát.** Trong học không giám sát, người ta không có mục tiêu cho nhiệm vụ muốn giải quyết, mà cố gắng nắm bắt 1 số đặc điểm của phân phối chung của các biến ngẫu nhiên quan sát được. Các nhiệm vụ đa dạng bao gồm ước lượng mật độ, giảm chiều, & phân cụm. Vì học không giám sát cho đến nay ít được chú ý kết hợp với tối ưu hóa tổ hợp & việc ứng dụng trực tiếp của nó có vẻ khó khăn, nên chúng tôi sẽ không thảo luận thêm về nó. Độc giả có thể tham khảo Bishop (2006); Goodfellow, Bengio, & Courville (2016); Murphy (2012) để biết thêm về sách giáo khoa về ML.

∗ Reinforcement learning. In reinforcement learning, an agent interacts with an environment through a Markov decision process, as illustrated in Fig. 2: Markov decision process associated with reinforcement learning, modified from Sutton & Barto (2018). Agent behavior is defined by its policy π, while environment evolution is defined by dynamics p. Note: reward is not necessary to define evolution & is provided only as a learning mechanism for agent. Actions, states, & rewards are random variables n general framework.. At every time step, agent is in a given state of environment & chooses an action according to its (possibly stochastic) policy. As a result, it receives from environment a reward & enters a new state. Goal in RL: train agent to maximize expected sum of future rewards, called *return*. For a given policy, expected return given a current state (resp., state & action pair) is known as value function (resp., state action value function). Value functions follow Bellman equation, hence problem can be formulated as dynamic programming, & solved approximately. Dynamics of environment need not be known by agent & are learned directly or indirectly, yielding an exploration vs. exploitation dilemma: choosing between between exploring new states for refining knowledge of environment for possible long-term improvements, or exploiting best-known scenario learned so far (which tends to be in already visited or predictable states).

– Học tăng cường. Trong học tăng cường, 1 tác nhân tương tác với môi trường thông qua 1 quy trình quyết định Markov, như minh họa trong Hình 2: Quy trình quyết định Markov liên quan đến học tăng cường, được sửa đổi từ Sutton & Barto (2018). Hành vi của tác nhân được xác định bởi chính sách π của nó, trong khi sự tiến hóa của môi trường được xác định bởi động lực p. Lưu ý: phần thưởng không cần thiết để xác định sự tiến hóa & chỉ được cung cấp như 1 cơ chế học tập cho tác nhân. Hành động, trạng thái, & phần thưởng là các biến ngẫu nhiên trong khuôn khổ chung.. Tại mỗi bước thời gian, tác nhân ở trong 1 trạng thái môi trường nhất định & chọn 1 hành động theo chính sách (có thể là ngẫu nhiên) của nó. Kết quả là, nó nhận được từ môi trường 1 phần thưởng & chuyển sang trạng thái mới. Mục tiêu trong RL: đào tạo tác nhân để tối đa hóa tổng kỳ vọng của các phần thưởng trong tương lai, được gọi là return. Đối với 1 chính sách nhất định, lợi nhuận kỳ vọng cho 1 trạng thái hiện tại (tương ứng là cặp trạng thái & hành động) được gọi là hàm giá trị (tương ứng là hàm giá trị trạng thái hành động). Các hàm giá trị tuân theo phương trình Bellman, do đó bài toán có thể được xây

dựng dưới dạng quy hoạch động, & giải gần đúng. Động lực của môi trường không cần phải được tác nhân biết & được học trực tiếp hoặc gián tiếp, dẫn đến tình huống khó xử giữa khám phá & khai thác: lựa chọn giữa việc khám phá các trạng thái mới để tinh chỉnh kiến thức về môi trường nhằm cải thiện khả năng dài hạn, hay khai thác kịch bản đã biết rõ nhất cho đến nay (thường ở trạng thái đã được biết đến hoặc có thể dự đoán được).
State should fully characterize environment at every step, in sense: future states only depend on past states via current state (Markov property). When this is not case, similar methods can be applied but we say: agent receives an *observation* of state. Markov property no longer 3.holds & Markov decision process is said to be partially observable.
– Trạng thái phải mô tả đầy đủ môi trường ở mỗi bước, theo nghĩa: trạng thái tương lai chỉ phụ thuộc vào trạng thái quá khứ thông qua trạng thái hiện tại (thuộc tính Markov). Khi không phải vậy, có thể áp dụng các phương pháp tương tự, nhưng chúng ta nói: tác nhân nhận được *quan sát* trạng thái. Thuộc tính Markov không còn 3. giữ nguyên & Quá trình quyết định Markov được cho là có thể quan sát 1 phần.
Defining a reward function is not always easy. Sometimes one would like to define a very sparse reward, e.g. 1 when agent solves problem, & 0 otherwise. Because of its underlying dynamic programming process, RL is naturally able to credit states/actions that lead to future rewards. Nonetheless, aforementioned setting is challenging as it provides no learning opportunity until agent (randomly, or through advanced approaches) solves problem. Furthermore, when policy is approximated (e.g., by a linear function), learning is not guaranteed to converge & may fall into local minima. E.g., an autonomous car may decide not to drive anywhere for fear of hitting a pedestrian & receiving a dramatic negative reward. These challenges are strongly related to aforementioned exploration dilemma. Reader is referred to to Sutton & Barto (2018) for an extensive textbook on RL.
– Việc xác định hàm phần thưởng không phải lúc nào cũng dễ dàng. Đôi khi, người ta muốn xác định 1 phần thưởng rất thưa thớt, ví dụ: 1 khi tác nhân giải quyết được vấn đề, & 0 nếu không. Do quy trình lập trình động cơ bản của nó, RL tự nhiên có thể ghi nhận các trạng thái/hành động dẫn đến phần thưởng trong tương lai. Tuy nhiên, việc thiết lập như đã đề cập ở trên rất khó khăn vì nó không cung cấp cơ hội học tập cho đến khi tác nhân (ngẫu nhiên hoặc thông qua các phương pháp nâng cao) giải quyết được vấn đề. Hơn nữa, khi chính sách được xấp xỉ (ví dụ: bằng hàm tuyến tính), việc học tập không được đảm bảo sẽ hội tụ & có thể rơi vào các cực tiểu cục bộ. Ví dụ: 1 chiếc xe tự hành có thể quyết định không lái xe đi đâu vì sợ đâm phải người đi bộ & nhận được phần thưởng tiêu cực đáng kể. Những thách thức này liên quan chặt chẽ đến tình thế tiến thoái lưỡng nan khám phá đã đề cập ở trên. Người đọc được giới thiệu đến Sutton & Barto (2018) để có 1 cuốn sách giáo khoa mở rộng về RL.

* Deep learning. DL is a successful method for building parametric composable functions in high dimensional spaces. In case of simplest neural network architecture, feedforward neural network (also called an multilayer perceptron), input data is successively passed through a number of layers. For every layer, an affine transformation is applied on input vector, followed by a nonlinear scalar function (named activation function) applied element-wise. Output of a layer, called *intermediate activations*, is passed on to next layer. All affine transformations are independent & represented in practice as different matrices of coefficients. They are learned, i.e., optimized over, through stochastic gradient descent, optimization algorithm used to minimize selected loss function. Stochasticity comes from limited number of data points used to compute loss before applying a gradient update. In practice, gradients are computed using reverse mode automatic differentiation, a practical algorithm based on chain rule, also known as back-propagation. Deep neural networks can be difficult to optimize, & a large variety of techniques have been developed to make optimization behave better, often by changing architectural designs of network. Because neural networks have dramatic capacities, i.e., they can essentially match any dataset, thus being prone to overfitting, they are also heavily regularized. Training them by stochastic gradient descent also regularizes them because of noise in gradient, making neural networks generally robust to overfitting issues, even when they are very large & would overfit if trained with more aggressive optimization. In addition, many hyper-parameters exist & different combinations are evaluated (known as hyper-parameters optimization). DL also sets itself apart from more traditional ML techniques by taking as inputs all available raw features of data, e.g., all pixels of an image, while traditional ML typically requires to engineer a limited number of domain-specific features.
– Học sâu (DL) là 1 phương pháp thành công để xây dựng các hàm hợp thành tham số trong không gian nhiều chiều. Trong trường hợp kiến trúc mạng nơ-ron đơn giản nhất, mạng nơ-ron truyền thẳng (còn gọi là perceptron đa lớp), dữ liệu đầu vào được truyền tuần tự qua 1 số lớp. Với mỗi lớp, 1 phép biến đổi afin được áp dụng trên vectơ đầu vào, tiếp theo là 1 hàm vô hướng phi tuyến tính (gọi là hàm kích hoạt) được áp dụng theo từng phần tử. Đầu ra của 1 lớp, được gọi là *intermediate activations*, được truyền sang lớp tiếp theo. Tất cả các phép biến đổi afin đều độc lập & trong thực tế được biểu diễn dưới dạng các ma trận hệ số khác nhau. Chúng được học, tức là được tối ưu hóa thông qua thuật toán tối ưu hóa giảm dần độ dốc ngẫu nhiên được sử dụng để tối thiểu hóa hàm mất mát đã chọn. Độ ngẫu nhiên đến từ số lượng điểm dữ liệu giới hạn được sử dụng để tính toán độ mất mát trước khi áp dụng cập nhật độ dốc. Trong thực tế, độ dốc được tính toán bằng cách sử dụng phép vi phân tự động chế độ ngược, 1 thuật toán thực tế dựa trên quy tắc chuỗi, còn được gọi là lan truyền ngược. Mạng nơ-ron sâu có thể khó tối ưu hóa, & nhiều kỹ thuật khác nhau đã được phát triển để tối ưu hóa hoạt động tốt hơn, thường bằng cách thay đổi thiết kế kiến trúc của mạng. Vì mạng nơ-ron có khả năng đáng kể, tức là về cơ bản chúng có thể khớp với bất kỳ tập dữ liệu nào, do đó dễ bị quá khớp, chúng cũng được chính quy hóa rất nhiều. Huấn luyện chúng bằng phương pháp giảm dần độ dốc ngẫu nhiên cũng chính quy hóa chúng do nhiễu trong độ dốc, khiến mạng nơ-ron nói chung mạnh mẽ trước các vấn đề quá khớp, ngay cả khi chúng rất lớn & sẽ quá khớp nếu được huấn luyện bằng tối ưu hóa mạnh mẽ hơn. Ngoài ra, có nhiều siêu tham số & các kết hợp khác nhau được đánh giá (được gọi là tối ưu hóa siêu tham số). DL cũng tự phân biệt mình với các kỹ thuật ML truyền thống hơn bằng cách lấy tất cả các đặc điểm thô có sẵn của dữ liệu làm đầu vào, ví dụ: tất cả các pixel của 1 hình ảnh, trong khi ML truyền thống thường yêu cầu thiết kế 1 số lượng hạn chế các đặc điểm cụ thể của miền.

DL researchers have developed different techniques to tackle this variety of structured data in a manner that can handle variable-size data structures, e.g., variable-length sequences. In this paragraph, & in next, present such state-of-art techniques. These are complex topics, but lack of comprehension does not hinder reading of paper. At a high level, it is enough to comprehend that these are architectures designed to handle different structures of data. Their usage, & in particular way they are learned, remains very similar to plain feedforward neural networks introduced above. 1st architectures presented are RNN. These models can operate on sequence data by *sharing parameters* across different sequence steps. More precisely, a same neural network block is successively applied at every step of sequence, i.e., with same architecture & parameter values at each time step. Specificity of such a network is presence of recurrent layers: layers that take as input both activation vector of previous layer & its own activation vector on preceding sequence step (called a *hidden state vector*), as illustrated in Fig. 3: A vanilla recurrent neural network modified from Goodfellow et al. (2016). On left, black square indicates a 1 step delay. On right, same RNN is shown unfolded. 3 sets U, V, W of parameters are represented & re-used at every time step.

– Các nhà nghiên cứu DL đã phát triển các kỹ thuật khác nhau để xử lý loại dữ liệu có cấu trúc này theo cách có thể xử lý các cấu trúc dữ liệu có kích thước thay đổi, ví dụ: các chuỗi có độ dài thay đổi. Trong đoạn này, & tiếp theo, trình bày các kỹ thuật tiên tiến như vậy. Đây là những chủ đề phức tạp, nhưng việc thiếu hiểu biết không cản trở việc đọc bài báo. Ở cấp độ cao, chỉ cần hiểu rằng đây là các kiến trúc được thiết kế để xử lý các cấu trúc dữ liệu khác nhau là đủ. Việc sử dụng chúng, & đặc biệt là cách chúng được học, vẫn rất giống với các mạng nơ-ron truyền thẳng thông thường được giới thiệu ở trên. Các kiến trúc đầu tiên được trình bày là RNN. Các mô hình này có thể hoạt động trên dữ liệu chuỗi bằng cách *chia sẻ các tham số* qua các bước chuỗi khác nhau. Chính xác hơn, 1 khối mạng nơ-ron giống nhau được áp dụng liên tiếp ở mọi bước của chuỗi, tức là với cùng 1 kiến trúc & các giá trị tham số tại mỗi bước thời gian. Điểm đặc trưng của 1 mạng như vậy là sự hiện diện của các lớp hồi quy: các lớp lấy đầu vào là cả vectơ kích hoạt của lớp trước đó & vectơ kích hoạt của chính nó ở bước trình tự trước đó (được gọi là *vector trạng thái ẩn*), như minh họa trong Hình 3: Một mạng nơ-ron hồi quy vanilla được sửa đổi từ Goodfellow & cộng sự (2016). Bên trái, hình vuông đen biểu thị độ trễ 1 bước. Bên phải, cùng 1 mạng nơ-ron nhân tạo được hiển thị ở dạng mở rộng. 3 tập tham số U, V, W được biểu diễn & tái sử dụng tại mỗi bước thời gian.

Another important size-invariant technique are *attention mechanisms*. They can be used to process data where each data point corresponds to a set. In that context, parameter sharing is used to address fact: different sets need not be of same size. Attention is used to query information about all elements in set, & merge it for downstream processing in neural network, as depicted in Fig. 4: A vanilla attention mechanism where a query q is computed against a set of values $(v_i)_i$. An affinity function f, e.g. a dot product, is used on query & value pairs. If it includes some parameters, mechanism can be learned. An affinity function takes as input query (which represents any kind of contextual information which informs where attention should be concentrated) & a representation of an element of set (both are activation vectors) & outputs a scalar. This is repeated over all elements in set for same query. Those scalars are normalized (e.g., with a softmax function) & used to define a weighted sum of representations of elements in set that can, in turn, be used in neural network making query. This form of content-based soft attention was introduced by Bahdanau, Cho, & Bengio (2015). A general explanation of attention mechanisms is given by Vaswani et al. (2017). Attention can be used to build graph neural network, i.e., neural networks able to process graph structured input data, as done by Veličković et al. (2018). In this architecture, every node attends over set of its neighbors. Process is repeated multiple times to gather information about nodes further away. GNN can also be understood as a form of message passing (Gilmer, Schoenholz, Riley, Vinyals, & Dahl, 2017).

– Một kỹ thuật bất biến kích thước quan trọng khác là *attention mechanisms*. Chúng có thể được sử dụng để xử lý dữ liệu trong đó mỗi điểm dữ liệu tương ứng với 1 tập hợp. Trong bối cảnh đó, chia sẻ tham số được sử dụng để giải quyết thực tế: các tập hợp khác nhau không nhất thiết phải có cùng kích thước. Attention được sử dụng để truy vấn thông tin về tất cả các phần tử trong tập hợp, & hợp nhất thông tin đó để xử lý xuôi dòng trong mạng nơ-ron, như được mô tả trong Hình 4: Một cơ chế chú ý đơn giản trong đó truy vấn q được tính toán dựa trên 1 tập hợp các giá trị $(v_i)_i$. Một hàm ái lực f, ví dụ: tích vô hướng, được sử dụng trên các cặp giá trị & truy vấn. Nếu nó bao gồm 1 số tham số, cơ chế có thể được học. Một hàm ái lực lấy đầu vào là truy vấn (biểu diễn bất kỳ loại thông tin ngữ cảnh nào cho biết nơi cần tập trung sự chú ý) & biểu diễn của 1 phần tử trong tập hợp (cả 2 đều là vectơ kích hoạt) & đầu ra là 1 số vô hướng. Điều này được lặp lại trên tất cả các phần tử trong tập hợp cho cùng 1 truy vấn. Các số vô hướng đó được chuẩn hóa (ví dụ: với hàm softmax) & được sử dụng để xác định tổng có trọng số của các biểu diễn của các phần tử trong tập hợp, sau đó có thể được sử dụng trong truy vấn tạo mạng nơ-ron. Hình thức chú ý mềm dựa trên nội dung này được giới thiệu bởi Bahdanau, Cho & Bengio (2015). Vaswani & cộng sự (2017) đưa ra lời giải thích chung về các cơ chế chú ý. Sự chú ý có thể được sử dụng để xây dựng mạng nơ-ron đồ thị, tức là các mạng nơ-ron có khả năng xử lý dữ liệu đầu vào có cấu trúc đồ thị, như đã được thực hiện bởi Veličković & cộng sự (2018). Trong kiến trúc này, mọi nút đều tham gia vào 1 tập hợp các nút lân cận của nó. Quá trình được lặp lại nhiều lần để thu thập thông tin về các nút ở xa hơn. GNN cũng có thể được hiểu là 1 hình thức truyền tin nhắn (Gilmer, Schoenholz, Riley, Vinyals, & Dahl, 2017).

DL & backpropagation can be used in supervised, unsupervised, or RL. Reader is referred to Goodfellow et al. (2016) for a ML textbook devoted to DL.

– DL & lan truyền ngược có thể được sử dụng trong học máy có giám sát, không giám sát hoặc học máy (RL). Độc giả có thể tham khảo Goodfellow & cộng sự (2016) để biết thêm về sách giáo khoa ML dành riêng cho DL.

- 3. Recent approaches. Survey different uses of ML to help solve combinatorial optimization problems & organize them along 2 orthogonal axes. 1st, in Sect. 3.1, illustrate 2 main motivations for using learning: approximation & discovery of new policies. Then, in Sect. 3.2, show examples of different ways to combine learned & traditional algorithmic elements.

– Khảo sát các ứng dụng khác nhau của ML để giúp giải quyết các bài toán tối ưu hóa tổ hợp & sắp xếp chúng theo 2 trục trực giao. 1. Trong Phần 3.1, minh họa 2 động lực chính của việc sử dụng học máy: xấp xỉ & khám phá các chính sách mới. Sau đó, trong Phần 3.2, trình bày các ví dụ về các cách khác nhau để kết hợp các yếu tố thuật toán đã học & truyền thống.

- 3.1. Learning methods. This sect relates to 2 motivations reported in Sect. 1.1 for using ML in combinatorial optimization. In some works, researcher assumes theoretical &/or empirical knowledge about decisions to be made for a combinatorial optimization algorithm, but wants to alleviate computational burden by approximating some of those decisions with ML. On contrary, we are also motivated by fact: sometimes, expert knowledge is not satisfactory & researcher wishes to find better ways of making decisions. Thus, ML can come into play to train a model through trial & error RL.

  – Phần này liên quan đến 2 động lực được báo cáo trong Phần 1.1 cho việc sử dụng ML trong tối ưu hóa tổ hợp. Trong 1 số công trình, nhà nghiên cứu giả định có kiến thức lý thuyết &/hoặc kinh nghiệm về các quyết định cần đưa ra cho 1 thuật toán tối ưu hóa tổ hợp, nhưng muốn giảm bớt gánh nặng tính toán bằng cách xấp xỉ 1 số quyết định đó với ML. Ngược lại, chúng tôi cũng được thúc đẩy bởi thực tế: đôi khi, kiến thức chuyên môn không thỏa đáng & nhà nghiên cứu muốn tìm ra những cách tốt hơn để đưa ra quyết định. Do đó, ML có thể được sử dụng để huấn luyện mô hình thông qua thử & sai RL.

  Frame both these motivations in state/action Markov decision process framework introduced in Sect. 2.2, where environment is internal state of algorithm. Care about learning algorithmic decisions utilized by a combinatorial optimization algorithm & call function making decision a *policy*, that, given all available information, [A *state* if information is sufficient to fully characterize environment at that time in a Markov decision process setting.] returns (possibly stochastically) action to be taken. Policy is function that we want to learn using ML & we show in following how 2 motivations naturally yield 2 learning settings. Note: case where length of trajectory of Markov decision process has value 1 is a common edge case (called *bandit setting*) where this formulation can seem excessive, but it nonetheless helps comparing methods.

  – Đóng khung cả 2 động cơ này trong khuôn khổ quy trình quyết định Markov hành động trạng thái được giới thiệu trong Phần 2.2, trong đó môi trường là trạng thái nội bộ của thuật toán. Hãy quan tâm đến việc học các quyết định thuật toán được sử dụng bởi thuật toán tối ưu hóa tổ hợp & gọi hàm đưa ra quyết định là *policy*, với tất cả thông tin có sẵn, [Một *state* nếu thông tin đủ để mô tả đầy đủ môi trường tại thời điểm đó trong bối cảnh quy trình quyết định Markov.] trả về (có thể là ngẫu nhiên) hành động cần thực hiện. Chính sách là hàm mà chúng ta muốn học bằng ML & chúng tôi trình bày trong phần sau cách 2 động cơ tự nhiên tạo ra 2 bối cảnh học. Lưu ý: trường hợp mà độ dài quỹ đạo của quy trình quyết định Markov có giá trị 1 là trường hợp ngoại lệ phổ biến (được gọi là *bandit setting*) trong đó công thức này có vẻ quá mức, nhưng dù sao nó vẫn hữu ích khi so sánh các phương pháp.

  In case of using ML to approximate decisions, policy is often learned by *imitation learning*, thanks to *demonstrations*, because expected behavior is shown (demonstrated) to ML model by an expert (also called *oracle*, even though it is not necessarily optimal), as shown in Fig. 5: In demonstration setting, policy is trained to reproduce action of an expert policy by minimizing some discrepancy in action space. In this setting, learner is not trained to optimize a performance measure, but to *blindly* mimic expert.

  – Trong trường hợp sử dụng ML để ước lượng các quyết định, chính sách thường được học bằng cách *học bắt chước*, nhờ vào *trình diễn*, vì hành vi mong đợi được 1 chuyên gia (còn gọi là *oracle*, mặc dù nó không nhất thiết là tối ưu) cho mô hình ML, như thể hiện trong Hình 5: Trong bối cảnh trình diễn, chính sách được đào tạo để tái tạo hành động của chính sách chuyên gia bằng cách giảm thiểu 1 số sự khác biệt trong không gian hành động. Trong bối cảnh này, người học không được đào tạo để tối ưu hóa thước đo hiệu suất, mà là để *bắt chước 1 cách mù quáng* chuyên gia.

  In case where one cares about discovering new policies, i.e., optimizing an algorithmic decision function from ground up, policy may be learned by RL through *experience*, as shown in Fig. 6: When learning through a reward signal, no expert is involved; only maximizing expected sum of future rewards (the return) matters.. Even though we present learning problem under fundamental Markov decision process of RL, this does not constrain one to use major RL algorithms (approximate dynamic programming & policy gradients) to maximize expected sum of rewards. Alternative optimization methods, e.g. bandit algorithms, genetic algorithms, direct/local search, can also be used to solve RL problem. [In general, identifying which algorithm will perform best is an open research question unlikely to have a simple answer, & is outside of scope of methodology presented here.]

  – Trong trường hợp quan tâm đến việc khám phá các chính sách mới, tức là tối ưu hóa hàm quyết định thuật toán từ đầu, thì chính sách có thể được RL học thông qua *experience*, như thể hiện trong Hình 6: Khi học thông qua tín hiệu phần thưởng, không có chuyên gia nào tham gia; chỉ có việc tối đa hóa tổng kỳ vọng của các phần thưởng trong tương lai (lợi nhuận) mới quan trọng.. Mặc dù chúng tôi trình bày vấn đề học tập theo quy trình quyết định Markov cơ bản của RL, nhưng điều này không ràng buộc người ta phải sử dụng các thuật toán RL chính (quy hoạch động xấp xỉ & gradient chính sách) để tối đa hóa tổng kỳ vọng của các phần thưởng. Các phương pháp tối ưu hóa thay thế, ví dụ như thuật toán bandit, thuật toán di truyền, tìm kiếm cục bộ trực tiếp, cũng có thể được sử dụng để giải quyết vấn đề RL. [Nhìn chung, việc xác định thuật toán nào sẽ hoạt động tốt nhất là 1 câu hỏi nghiên cứu mở khó có thể có câu trả lời đơn giản, & nằm ngoài phạm vi phương pháp luận được trình bày ở đây.]

  Critical to understand: in imitation setting, policy is learned through supervised targets provided by an expert for every action (& without a reward), whereas in experience setting, policy is learned from a reward (possibly delayed) signal using RL (& without an expert). In imitation, agent is taught *what* to do, whereas in RL, agent is encouraged to quickly *accumulate* rewards. Distinction between these 2 settings is far more complex than distinction made here. Explore some of this complexity, including their strengths & weaknesses, in Sect. 5.1.

  – Điều quan trọng cần hiểu: trong bối cảnh bắt chước, chính sách được học thông qua các mục tiêu được giám sát do chuyên gia cung cấp cho mỗi hành động (không có phần thưởng), trong khi trong bối cảnh trải nghiệm, chính sách được học từ tín

hiệu phần thưởng (có thể bị trì hoãn) bằng cách sử dụng Thực tế tăng cường (không có chuyên gia). Trong bối cảnh bắt chước, tác nhân được dạy phải làm gì, trong khi trong Thực tế tăng cường, tác nhân được khuyến khích nhanh chóng tích lũy phần thưởng. Việc phân biệt giữa 2 bối cảnh này phức tạp hơn nhiều so với sự phân biệt được thực hiện ở đây. Hãy khám phá 1 số sự phức tạp này, bao gồm cả điểm mạnh & điểm yếu của chúng, trong Phần 5.1.

In following, few papers demonstrating different settings are surveyed.

* 3.1.1. Demonstration. In Baltean-Lugojan, Misener, Bonami, & Tramontani (2018), authors use a neural network to approximate lower bound improvement generated by tightening current relaxation via cutting planes (cuts, for short). More precisely, Baltean-Lugojan et al. (2018) consider non-convex quadratic programming problems & aim at approximating associated semidefinite programming relaxation, known to be strong but time-consuming, by a linear program. A straightforward way of doing that: iteratively add (linear) cutting planes associated with negative eigenvalues, especially considering small-size (square) submatrices of original quadratic objective function. That approach has advantage of generating sparse cuts [Reader is referred to Dey & Molinaro (2018) for a detailed discussion on importance of sparse cutting planes in mixed-integer linear programming.] but it is computationally challenging because of exponential number of those submatrices & because of difficulty of finding right metrics to select among violated cuts. Authors propose to solve semidefinite programming to compute bound improvement associated with considering specific submatrices, which is also a proxy on equality of cuts that could be separated from same submatrices. In this context, supervised (imitation) learning is applied offline to approximate objective value of semidefinite programming problem associated with a submatrix selection &, afterward, model can be rapidly applied to select most promising submatrices without very significant computational burden of solving semidefinite programming. Of course, rational: most promising submatrices corresponding to most promising cutting planes & Baltean-Lugojan et al. (2018) train a model to estimate objective of an semidefinite programming problem only in order to decide to add most promising cutting planes. Hence, cutting plane selection is ultimate policy learned.

  – Trong Baltean-Lugojan, Misener, Bonami, & Tramontani (2018), các tác giả sử dụng mạng nơ-ron để xấp xỉ sự cải thiện giới hạn dưới được tạo ra bằng cách thắt chặt sự giãn dòng điện thông qua các mặt phẳng cắt (gọi tắt là các vết cắt). Chính xác hơn, Baltean-Lugojan & cộng sự (2018) xem xét các bài toán lập trình bậc 2 không lồi & hướng tới việc xấp xỉ sự giãn lập trình bán xác định liên quan, được biết là mạnh nhưng tốn thời gian, bằng 1 chương trình tuyến tính. Một cách đơn giản để thực hiện điều đó: lặp lại việc cộng các mặt phẳng cắt (tuyến tính) liên quan đến các giá trị riêng âm, đặc biệt khi xem xét các ma trận con (vuông) có kích thước nhỏ của hàm mục tiêu bậc 2 ban đầu. Cách tiếp cận đó có ưu điểm là tạo ra các vết cắt thưa thớt [Người đọc được giới thiệu Dey & Molinaro (2018) để thảo luận chi tiết về tầm quan trọng của các mặt phẳng cắt thưa thớt trong lập trình tuyến tính số nguyên hỗn hợp.] nhưng về mặt tính toán, nó rất khó khăn do số lượng theo cấp số mũ của các ma trận con đó & do khó tìm đúng số liệu để lựa chọn trong số các vết cắt bị vi phạm. Các tác giả đề xuất giải quyết lập trình bán xác định để tính toán cải tiến ràng buộc liên quan đến việc xem xét các ma trận con cụ thể, đây cũng là 1 phép biến đổi về tính bằng nhau của các phép cắt có thể tách ra từ cùng các ma trận con. Trong bối cảnh này, học có giám sát (mô phỏng) được áp dụng ngoại tuyến để xấp xỉ giá trị mục tiêu của bài toán lập trình bán xác định liên quan đến việc lựa chọn ma trận con &, sau đó, mô hình có thể được áp dụng nhanh chóng để lựa chọn các ma trận con hứa hẹn nhất mà không cần gánh nặng tính toán đáng kể khi giải quyết lập trình bán xác định. Tất nhiên, hợp lý: các ma trận con hứa hẹn nhất tương ứng với các mặt phẳng cắt hứa hẹn nhất & Baltean-Lugojan & cộng sự (2018) đã huấn luyện 1 mô hình để ước tính mục tiêu của 1 bài toán lập trình bán xác định chỉ để quyết định thêm các mặt phẳng cắt hứa hẹn nhất. Do đó, lựa chọn mặt phẳng cắt là chính sách cuối cùng đã học được.

  Another example of demonstration is found in context of branching policies in branch-&-bound trees of mixed-integer linear programming. Choice of variables to branch on can dramatically change size of branch-&-bound tree, hence solving time. Among many heuristics, a well-performing approach is *strong branching* (Applegate, Bixby, Chvátal, Cook, 2007). Namely, for every branching decision to be made, strong branching performs a 1 step look-ahead by tentatively branching on many candidate variables, computes linear programming relaxations to get potential lower bound improvements, & eventually branches on variable providing best improvement. Even if not all variables are explored, & linear programming value can be approximated, this is still a computationally expensive strategy. For these reasons, Marcos Alvarez, Louveaux, & Wehenkel (2014, 2017) use a special type of decision tree (a classical model supervised learning) to approximate strong branching decisions using supervised learning. Khalil, Bodic, Song, Nemhauser, & Dilkina (2016) propose a similar approach, where a linear model is learned on fly for every instance by using strong branching at top of tree, & eventually replacing it by its ML approximation. Linear approximator of strong branching introduced in Marcos Alvarez, Wehenkel, & Louveaux (2016) is learned in an active fashion: when estimator is deemed unreliable, algorithm falls back to true strong branching & results are then used for both branching & learning. In all branching algorithms presented here, inputs to ML model are engineered as a vector of fixed length with static features descriptive of instance, & dynamic features providing information about state of branch-&-bound process. Gasse, Chételat, Ferroni, Charlin, & Lodi (2019) use a neural network to learn an offine approximation to strong branching, but, contrary to aforementioned papers, authors use a raw exhaustive representation (i.e., they do not discard nor aggregate any information) of sub-problem associated with current branching node as input to ML model. Namely, an mixed-integer linear programming subproblem is represented as a bipartite graph on variables & constraints, with edges representing nonzero coefficients in constraint matrix. Each node is augmented with a set of features to fully describe sub-problem, & a GNN is used to build an ML approximator able to process this type of structured data. Node selection, i.e., deciding on next branching node to explore in a branch-&-bound tree, is also a critical decision in mixed-integer linear programming. He, Daume III, & Eisner (2014) learn a policy to select among open branching nodes the one that contains optimal solution in its subtree. Training algorithm is an online

learning method collecting expert behaviors throughout entire learning phase. Reader is referred to Lodi & Zarpellon (2017) for an extended survey on learning & branching in mixed-integer linear programming.

– Một ví dụ minh họa khác được tìm thấy trong bối cảnh các chính sách phân nhánh trong cây nhánh-&-bound của lập trình tuyến tính số nguyên hỗn hợp. Việc lựa chọn các biến để phân nhánh có thể thay đổi đáng kể kích thước của cây nhánh-&-bound, do đó làm tăng thời gian giải. Trong số nhiều phương pháp tìm kiếm, 1 cách tiếp cận hiệu quả là *strong branching* (Applegate, Bixby, Chvátal, Cook, 2007). Cụ thể, đối với mỗi quyết định phân nhánh được đưa ra, phân nhánh mạnh thực hiện xem trước 1 bước bằng cách phân nhánh thử nghiệm trên nhiều biến ứng viên, tính toán các mức nới lỏng lập trình tuyến tính để có được các cải tiến cận dưới tiềm năng, & cuối cùng phân nhánh trên biến cung cấp cải tiến tốt nhất. Ngay cả khi không phải tất cả các biến đều được khám phá, & giá trị lập trình tuyến tính có thể được xấp xỉ, thì đây vẫn là 1 chiến lược tốn kém về mặt tính toán. Vì những lý do này, Marcos Alvarez, Louveaux, & Wehenkel (2014, 2017) sử dụng 1 loại cây quyết định đặc biệt (một mô hình học có giám sát cổ điển) để xấp xỉ các quyết định phân nhánh mạnh bằng cách sử dụng học có giám sát. Khalil, Bodic, Song, Nemhauser & Dilkina (2016) đề xuất 1 phương pháp tương tự, trong đó 1 mô hình tuyến tính được học trực tiếp cho mọi trường hợp bằng cách sử dụng phân nhánh mạnh ở đỉnh cây, & cuối cùng thay thế nó bằng phép xấp xỉ ML của nó. Xấp xỉ tuyến tính của phân nhánh mạnh được giới thiệu trong Marcos Alvarez, Wehenkel & Louveaux (2016) được học theo cách chủ động: khi ước lượng được coi là không đáng tin cậy, thuật toán sẽ quay lại phân nhánh mạnh thực sự & kết quả sau đó được sử dụng cho cả phân nhánh & học. Trong tất cả các thuật toán phân nhánh được trình bày ở đây, đầu vào cho mô hình ML được thiết kế dưới dạng 1 vectơ có độ dài cố định với các đặc trưng tĩnh mô tả trường hợp, & các đặc trưng động cung cấp thông tin về trạng thái của tiến trình ràng buộc nhánh. Gasse, Chételat, Ferroni, Charlin, & Lodi (2019) sử dụng mạng nơ-ron để học xấp xỉ ngoại tuyến cho nhánh mạnh, nhưng trái ngược với các bài báo đã đề cập, các tác giả sử dụng biểu diễn đầy đủ thô (tức là họ không loại bỏ hoặc tổng hợp bất kỳ thông tin nào) của bài toán con liên quan đến nút nhánh hiện tại làm đầu vào cho mô hình ML. Cụ thể, 1 bài toán con lập trình tuyến tính số nguyên hỗn hợp được biểu diễn dưới dạng đồ thị 2 phía trên các biến & ràng buộc, với các cạnh biểu diễn các hệ số khác không trong ma trận ràng buộc. Mỗi nút được bổ sung 1 tập hợp các đặc trưng để mô tả đầy đủ bài toán con, & 1 GNN được sử dụng để xây dựng 1 bộ xấp xỉ ML có khả năng xử lý loại dữ liệu có cấu trúc này. Việc lựa chọn nút, tức là quyết định nút nhánh tiếp theo để khám phá trong 1 cây nhánh &-bound, cũng là 1 quyết định quan trọng trong lập trình tuyến tính số nguyên hỗn hợp. He, Daume III, & Eisner (2014) tìm hiểu 1 chính sách để chọn trong số các nút nhánh mở, nút chứa giải pháp tối ưu trong cây con của nó. Thuật toán huấn luyện là 1 phương pháp học trực tuyến, thu thập các hành vi của chuyên gia trong suốt quá trình học. Độc giả có thể tham khảo Lodi & Zarpellon (2017) để biết thêm về khảo sát mở rộng về học & phân nhánh trong quy hoạch tuyến tính số nguyên hỗn hợp.

Branch & bound is a technique not limited to mixed-integer linear programming & can be use for general tree search. Hottung, Tanaka, & Tierney (2017) build a tree search procedure for container pre-marshalling problem in which they aim to learn, not only a branching policy (similar in principle to what has been discussed in previous paragraph), but also a value network to estimate value of partial solutions & used for bounding. Authors leverage a form of CNN [A type of neural network, usually used on image input, that leverages parameter sharing to extract local information.] .for both networks & train them in a supervised fashion using pre-computed solutions of problem. Resulting algorithm is heuristic due to approximations made while bounding.

– Branch & bound là 1 kỹ thuật không chỉ giới hạn ở quy hoạch tuyến tính số nguyên hỗn hợp & có thể được sử dụng cho tìm kiếm cây tổng quát. Hottung, Tanaka, & Tierney (2017) xây dựng 1 quy trình tìm kiếm cây cho bài toán tiền sắp xếp container, trong đó họ hướng đến việc học không chỉ 1 chính sách phân nhánh (tương tự về nguyên tắc với những gì đã thảo luận ở đoạn trước), mà còn 1 mạng giá trị để ước tính giá trị của các nghiệm riêng phần & được sử dụng cho việc bao hàm. Các tác giả sử dụng 1 dạng CNN [Một loại mạng nơ-ron, thường được sử dụng cho đầu vào hình ảnh, tận dụng việc chia sẻ tham số để trích xuất thông tin cục bộ.] cho cả 2 mạng & huấn luyện chúng theo cách có giám sát bằng cách sử dụng các nghiệm được tính toán trước của bài toán. Thuật toán kết quả mang tính chất heuristic do các phép xấp xỉ được thực hiện trong khi bao hàm.

Learning a policy by demonstration is identical to supervised learning, where training pairs of input state & target actions are provided by expert. In simplest case, expert decisions are collected beforehand, but more advanced methods can collect them online to increase stability as previously shown in Marcos Alvarez et al. (2016) & He et al. (2014).

– Học chính sách bằng cách trình diễn giống hệt với học có giám sát, trong đó các cặp huấn luyện trạng thái đầu vào & hành động mục tiêu được cung cấp bởi chuyên gia. Trong trường hợp đơn giản nhất, các quyết định của chuyên gia được thu thập trước, nhưng các phương pháp tiên tiến hơn có thể thu thập chúng trực tuyến để tăng tính ổn định như đã được trình bày trước đây trong Marcos Alvarez & cộng sự (2016) & He & cộng sự (2014).

∗ 3.1.2. Experience. Considering TSP on a graph, easy to devise a greedy heuristic that builds a tour by sequentially picking nodes among these that have not been visited yet, hence defining a permutation. If criterion for selecting next node is to take closest one, then heuristic is known as nearest neighbor. This simple heuristic has poor practical performance & many other heuristics perform better empirically, i.e., build cheaper tours. Selecting nearest node is a fair intuition but turns out to be a far from satisfactory. Khalil, Dai, Zhang, Dilkina, & Song (2017a) suggest learning criterion for this selection. They build a greedy heuristic framework, where node selection policy is learned using a GNN (Dai, Dai, & Song, 2016), a type of neural network able to process input graphs of any finite size by a mechanism of message passing (Gilmer et al., 2017). For every node to select, authors feed to network graph representation of problem – augmented with features indicating which of nodes have already been visited – & receive back an action value for every node. Action values are used to train network through RL (Q-learning in particular) & partial tour length is used as a reward.

– Xét TSP trên đồ thị, dễ dàng thiết kế 1 phương pháp heuristic tham lam xây dựng 1 hành trình bằng cách chọn tuần

tự các nút trong số các nút chưa được ghé thăm, do đó xác định 1 hoán vị. Nếu tiêu chí để chọn nút tiếp theo là lấy nút gần nhất, thì phương pháp heuristic được gọi là láng giềng gần nhất. Phương pháp heuristic đơn giản này có hiệu suất thực tế kém & nhiều phương pháp heuristic khác hoạt động tốt hơn theo kinh nghiệm, tức là xây dựng các hành trình rẻ hơn. Việc chọn nút gần nhất là 1 trực giác công bằng nhưng hóa ra lại không thỏa đáng. Khalil, Dai, Zhang, Dilkina, & Song (2017a) đề xuất tiêu chí học cho lựa chọn này. Họ xây dựng 1 khuôn khổ heuristic tham lam, trong đó chính sách lựa chọn nút được học bằng cách sử dụng GNN (Dai, Dai, & Song, 2016), 1 loại mạng nơ-ron có khả năng xử lý đồ thị đầu vào có kích thước hữu hạn bất kỳ bằng cơ chế truyền thông điệp (Gilmer & cộng sự, 2017). Với mỗi nút được chọn, tác giả sẽ đưa vào biểu diễn đồ thị mạng về vấn đề – được bổ sung các đặc điểm cho biết nút nào đã được truy cập – & nhận lại giá trị hành động cho mỗi nút. Giá trị hành động được sử dụng để huấn luyện mạng thông qua RL (đặc biệt là Q-learning) & 1 phần chiều dài chuyến tham quan được sử dụng làm phần thưởng.

This example does not do justice to rich TSP literature that has developed far more advanced algorithms performing orders of magnitude better than ML ones. Nevertheless, point we are trying to highlight here: given a fixed context, & a decision to be made, ML can be used to discover new, potentially better performing policies. Even on state-of-art TSP algorithms (i.e., when exact solving is taken to its limits), many decisions are made in heuristic ways, e.g., cutting plane selection, thus leaving room for ML to assist in making these decisions.

– Ví dụ này không thể hiện hết được sự phong phú của các tài liệu TSP, vốn đã phát triển các thuật toán tiên tiến hơn nhiều, hoạt động tốt hơn gấp bội so với ML. Tuy nhiên, điểm chúng tôi muốn nhấn mạnh ở đây: với 1 bối cảnh cố định, & 1 quyết định cần đưa ra, ML có thể được sử dụng để khám phá các chính sách mới, có khả năng hoạt động tốt hơn. Ngay cả trên các thuật toán TSP tiên tiến (tức là khi việc giải chính xác đạt đến giới hạn của nó), nhiều quyết định vẫn được đưa ra theo phương pháp heuristic, ví dụ như lựa chọn mặt phẳng cắt, do đó tạo điều kiện cho ML hỗ trợ việc đưa ra các quyết định này.

Once again, stress: learning a policy by experience is well described by Markov decision process framework of RL, where an agent maximizes return (defined in Sect. 2.2). By matching reward signal with optimization objective, goal of learning agent becomes to solve problem, without assuming any expert knowledge. Some methods that were not presented as RL can also be cast in this Markov decision process formulation, even if optimization methods are not those of RL community. E.g., part of combinatorial optimization literature is dedicated to automatically build specialized heuristics for different problems. Heuristics are built by orchestrating a set of moves, or subroutines, from a pre-defined domain-specific collections. E.g., to tackle bipartite boolean quadratic programming problems, Karapetyan, Punnen, & Parkes (2017) represent this orchestration as a Markov chain where states are subroutines. 1 Markov chain is parameterized by its transition probabilities. Mascia, López-Ibáñez, Dubois-Lacoste, & Stützle (2014), on other hand, define valid succession of moves through a grammar, where words are moves & sentences correspond to heuristics. Authors introduce a parametric space to represent sentences of a grammar. In both cases, setting is very close to Markov decision process of RL, but parameters are learned though direct optimization of performances of their associated heuristic through so-called *automatic configuration tools* (usually based on genetic or local search, & exploiting parallel computing). Note: learning setting is rather simple as parameters do not adapt to problem instance, but are fixed for various clusters. From ML point of view, this is equivalent to a piece-wise constant regression. If more complex models were to be used, direct optimization may not scale adequately to obtain good performances. Same approach to building heuristics can be brought 1 level up if, instead of orchestrating sets of moves, it arranges predefined heuristics. Resulting heuristic is then called a *hyper-heuristic*. Özcan, Misir, Ochoa, & Burke (2012) build a hyper-heuristic for examination timetabling by learning to combine existing heuristics. They use a bandit algorithm, a stateless form of RL (see Sutton & Barto, 2018, Chap. 2), to learn online a value function for each heuristic.

– Một lần nữa, nhấn mạnh: việc học 1 chính sách bằng kinh nghiệm được mô tả rõ ràng trong khuôn khổ quy trình quyết định Markov của RL, trong đó 1 tác nhân tối đa hóa lợi nhuận (được định nghĩa trong Phần 2.2). Bằng cách kết hợp tín hiệu phần thưởng với mục tiêu tối ưu hóa, mục tiêu của tác nhân học trở thành giải quyết vấn đề mà không cần giả định bất kỳ kiến thức chuyên môn nào. Một số phương pháp không được trình bày dưới dạng RL cũng có thể được đưa vào công thức quy trình quyết định Markov này, ngay cả khi các phương pháp tối ưu hóa không phải là phương pháp của cộng đồng RL. Ví dụ: 1 phần của tài liệu tối ưu hóa tổ hợp được dành riêng để tự động xây dựng các phương pháp tìm kiếm chuyên biệt cho các vấn đề khác nhau. Các phương pháp tìm kiếm được xây dựng bằng cách phối hợp 1 tập hợp các bước di chuyển hoặc các chương trình con từ các tập hợp cụ thể theo miền được xác định trước. Ví dụ: để giải quyết các vấn đề lập trình bậc 2 boolean 2 phần, Karapetyan, Punnen & Parkes (2017) biểu diễn sự phối hợp này dưới dạng chuỗi Markov trong đó các trạng thái là các chương trình con. 1 Chuỗi Markov được tham số hóa theo xác suất chuyển tiếp của nó. Mặt khác, Mascia, López-Ibáñez, Dubois-Lacoste, & Stützle (2014) định nghĩa chuỗi di chuyển hợp lệ trong 1 ngữ pháp, trong đó các từ là các di chuyển & các câu tương ứng với các phương pháp tìm kiếm. Các tác giả giới thiệu 1 không gian tham số để biểu diễn các câu của 1 ngữ pháp. Trong cả 2 trường hợp, thiết lập rất gần với quy trình quyết định Markov của RL, nhưng các tham số được học thông qua tối ưu hóa trực tiếp hiệu suất của phương pháp tìm kiếm liên quan thông qua cái gọi là *các công cụ cấu hình tự động* (thường dựa trên tìm kiếm di truyền hoặc cục bộ, & khai thác điện toán song song). Lưu ý: việc học thiết lập khá đơn giản vì các tham số không thích ứng với trường hợp bài toán mà được cố định cho các cụm khác nhau. Theo quan điểm của ML, điều này tương đương với hồi quy hằng số từng phần. Nếu sử dụng các mô hình phức tạp hơn, tối ưu hóa trực tiếp có thể không mở rộng đủ để đạt được hiệu suất tốt. Cách tiếp cận tương tự để xây dựng phương pháp tìm kiếm có thể được nâng lên 1 cấp nếu, thay vì sắp xếp các tập hợp các bước di chuyển, nó sắp xếp các phương pháp tìm kiếm được xác định trước. Phương pháp heuristic kết quả sau đó được gọi là *hyper-heuristic*. Özcan, Misir, Ochoa & Burke (2012) xây dựng 1 phương pháp heuristic siêu hạng để lập thời khóa biểu thi cử bằng cách học cách kết hợp các phương pháp heuristic hiện có. Họ sử dụng thuật toán bandit, 1 dạng RL không trạng thái (xem Sutton & Barto, 2018, Chương 2), để học trực tuyến 1 hàm giá trị cho mỗi phương pháp heuristic.

Close this sect by noting: demonstration & experience are not mutually exclusive & most learning tasks can be tackled in both ways. In case of selecting branching variables in an mixed-integer linear programming branch-&-bound tree, one could adopt anyone of 2 prior strategies. On 1 hand, Khalil et al. (2016); Marcos Alvarez et al. (2014, 2017); Marcos Alvarez et al. (2016) estimate: strong branching is an effective branching strategy but computationally too expensive & build a ML model to approximate it. On other hand, one could believe: no branching strategy is good enough & try to learn one from ground up, e.g. through RL as suggested (but not implemented) in Khalil et al. (2016). An intermediary approach is proposed by Liberto, Kadioglu, Leo, & Malitsky (2016). Authors recognize: among traditional variable selection policies, the ones performing well at top of branch-&-bound tree are not necessarily same as ones performing well deeper down. Hence, authors learn a model to dynamically switch among predefined policies during branch-&-bound based on current state of tree. While this seems like a case of imitation learning, given that traditional branching policies can be thought of as experts, this is actually not the case. In fact, model is not learning *from* any expert, but really learning to choose between pre-existing policies. This is technically not a branching variable selection, but rather a branching heuristic selection policy. Each sub-tree is represented by a vector of handcrafted features, & a clustering of these vectors is performed. Similarly to what was detailed in previous paragraph about the work of Karapetyan et al. (2017); Mascia et al. (2014), automatic configuration tools are then used to assign best branching policy to each cluster. When branching at a given node, cluster closest to current sub-tree is retrieved, & its assigned policy is used.

– Kết thúc phần này bằng cách lưu ý: trình diễn & kinh nghiệm không loại trừ lẫn nhau & hầu hết các nhiệm vụ học tập có thể được giải quyết theo cả 2 cách. Trong trường hợp chọn các biến phân nhánh trong cây ràng buộc nhánh lập trình tuyến tính số nguyên hỗn hợp, người ta có thể áp dụng bất kỳ chiến lược nào trong 2 chiến lược trước đó. Một mặt, Khalil & cộng sự (2016); Marcos Alvarez & cộng sự (2014, 2017); Marcos Alvarez & cộng sự (2016) ước tính: phân nhánh mạnh là 1 chiến lược phân nhánh hiệu quả nhưng quá tốn kém về mặt tính toán & xây dựng 1 mô hình ML để xấp xỉ nó. Mặt khác, người ta có thể tin rằng: không có chiến lược phân nhánh nào là đủ tốt & cố gắng học 1 chiến lược từ đầu, ví dụ thông qua RL như được đề xuất (nhưng không được triển khai) trong Khalil & cộng sự (2016). Một cách tiếp cận trung gian được Liberto, Kadioglu, Leo, & Malitsky (2016) đề xuất. Các tác giả nhận thấy: trong số các chính sách lựa chọn biến truyền thống, những chính sách hoạt động tốt ở đầu cây nhánh-&-bound không nhất thiết giống với những chính sách hoạt động tốt ở sâu hơn. Do đó, các tác giả học 1 mô hình để chuyển đổi động giữa các chính sách được xác định trước trong nhánh-&-bound dựa trên trạng thái hiện tại của cây. Mặc dù điều này có vẻ giống như 1 trường hợp học bắt chước, vì các chính sách phân nhánh truyền thống có thể được coi là chuyên gia, nhưng thực tế không phải vậy. Trên thực tế, mô hình không học *từ* bất kỳ chuyên gia nào, mà thực sự học cách lựa chọn giữa các chính sách đã có từ trước. Về mặt kỹ thuật, đây không phải là lựa chọn biến phân nhánh, mà là 1 chính sách lựa chọn heuristic phân nhánh. Mỗi cây con được biểu diễn bằng 1 vectơ các đặc trưng thủ công, & việc phân cụm các vectơ này được thực hiện. Tương tự như những gì đã được trình bày chi tiết trong đoạn trước về công trình của Karapetyan & cộng sự (2017); Mascia & cộng sự (2014), các công cụ cấu hình tự động sau đó được sử dụng để chỉ định chính sách phân nhánh tốt nhất cho mỗi cụm. Khi phân nhánh tại 1 nút nhất định, cụm gần nhất với cây con hiện tại sẽ được truy xuất & chính sách được chỉ định của cụm đó sẽ được sử dụng.

- 3.2. Algorithmic structure. In this sect, survey how learned policies (whether from demonstration or experience) are combined with traditional combinatorial optimization algorithms, i.e., considering ML & explicit algorithms as building blocks, survey how they can be laid out in different templates. 3 following sects are not necessarily disjoint nor exhaustive but are a natural way to look at literature.

  – Cấu trúc thuật toán. Trong phần này, hãy khảo sát cách các chính sách đã học (dù từ trình diễn hay kinh nghiệm) được kết hợp với các thuật toán tối ưu hóa tổ hợp truyền thống, tức là coi ML & thuật toán rõ ràng là các khối xây dựng, hãy khảo sát cách chúng có thể được trình bày trong các mẫu khác nhau. 3 phần sau không nhất thiết phải rời rạc hay đầy đủ nhưng là 1 cách tự nhiên để xem xét tài liệu.

  * 3.2.1. End to end learning. A 1st idea ot leverage ML to solve discrete optimization problems: train ML model to output solutions directly from input instance, as shown in Fig. 7: ML acts alone to provide a solution to problem.

    – Học tập từ đầu đến cuối. Ý tưởng đầu tiên để tận dụng ML để giải quyết các vấn đề tối ưu hóa rời rạc: đào tạo mô hình ML để đưa ra giải pháp trực tiếp từ trường hợp đầu vào, như thể hiện trong Hình 7: ML hoạt động độc lập để cung cấp giải pháp cho vấn đề.

    This approach has been explored recently, especially on Euclidean TSP. To tackle problem with DL, Vinyals, Fortunato, & Jaitly (2015) introduce pointer network wherein an encoder, namely an RNN, is used to parse all TSP nodes in input graph & produces an encoding (a vector of activations) for each of them. Afterward, a decoder, also an RNN, uses an attention mechanism similar to Bahdanau et al. (2015) (Sect. 2.2) over perviously encoded nodes in graph to produce a probability distribution over these nodes (through softmax layer previously illustrated in Fig. 4: A vanilla attention mechanism where a query q is computed against a set of values $(v_i)_i$. An affinity function f, e.g. a dot product, is used on query & value pairs. If it includes some parameters, mechanism can be learned.) Repeating this decoding step, possible for network to output a permutation over its inputs (TSP nodes). This method makes it possible to use network over different input graph sizes. Authors train model through supervised learning with precomputed TSP solutions as targets. Bello, Pham, Le, Norouzi, & Bengio (2017) use a similar model & train it with RL using tour length as a reward signal. They address some limitations of supervised (imitation) learning, e.g. need to compute optimal (or at least high quality) TSP solutions (targets), that in turn, may be ill-defined when those solutions are not computed exactly, or when multiple solutions exist. Kool & Welling (2018) introduce more prior knowledge in model using a GNN instead of an RNN to process input. Emami & Ranka (2018) & Nowak, Villar, Bandeira, & Bruna (2017) explore a different approach by directly approximating a double stochastic matrix in output of neural network to characterize permutation. Work of Khalil et al. (2017a), introduced in Sect. 3.1.2,

can also be understood as an end to end method to tackle TSP, but prefer to see it under eye of Sect. 3.2.3. Worth noting: tackling TSP through ML is not new. Earlier work from 1990s focused on Hopfield neural networks & self organizing neural networks, interested reader is referred to survey of Smith (1999).
– Cách tiếp cận này đã được khám phá gần đây, đặc biệt là trên TSP Euclidean. Để giải quyết vấn đề với DL, Vinyals, Fortunato & Jaitly (2015) đã giới thiệu mạng con trỏ, trong đó 1 bộ mã hóa, cụ thể là RNN, được sử dụng để phân tích tất cả các nút TSP trong đồ thị đầu vào & tạo ra 1 mã hóa (một vectơ kích hoạt) cho mỗi nút. Sau đó, 1 bộ giải mã, cũng là RNN, sử dụng cơ chế chú ý tương tự như Bahdanau & cộng sự. (2015) (Phần 2.2) trên các nút được mã hóa trước đó trong đồ thị để tạo ra phân phối xác suất trên các nút này (thông qua lớp softmax đã minh họa trước đó trong Hình 4: Cơ chế chú ý vani trong đó truy vấn q được tính toán dựa trên tập hợp các giá trị $(v_i)_i$. Hàm ái lực f, ví dụ: tích vô hướng, được sử dụng trên các cặp giá trị truy vấn &. Nếu nó bao gồm 1 số tham số, cơ chế có thể được học.) Lặp lại bước giải mã này, mạng có thể đưa ra 1 hoán vị trên các đầu vào của nó (các nút TSP). Phương pháp này giúp có thể sử dụng mạng trên các kích thước đồ thị đầu vào khác nhau. Các tác giả đào tạo mô hình thông qua học có giám sát với các giải pháp TSP được tính toán trước làm mục tiêu. Bello, Pham, Le, Norouzi, & Bengio (2017) sử dụng 1 mô hình tương tự & đào tạo nó bằng RL sử dụng độ dài chuyến tham quan làm tín hiệu thưởng. Họ giải quyết 1 số hạn chế của học có giám sát (mô phỏng), ví dụ: cần tính toán các giải pháp TSP tối ưu (hoặc ít nhất là chất lượng cao) (mục tiêu), đến lượt nó, có thể không được xác định rõ ràng khi các giải pháp đó không được tính toán chính xác hoặc khi tồn tại nhiều giải pháp. Kool & Welling (2018) giới thiệu thêm kiến thức trước đó trong mô hình sử dụng GNN thay vì RNN để xử lý đầu vào. Emami & Ranka (2018) & Nowak, Villar, Bandeira & Bruna (2017) khám phá 1 cách tiếp cận khác bằng cách trực tiếp xấp xỉ ma trận ngẫu nhiên kép trong đầu ra của mạng nơ-ron để mô tả hoán vị. Công trình của Khalil & cộng sự (2017a), được giới thiệu trong Phần 3.1.2, cũng có thể được hiểu là 1 phương pháp đầu cuối để giải quyết TSP, nhưng muốn xem xét nó dưới góc nhìn của Phần 3.2.3. Cần lưu ý: việc giải quyết TSP thông qua ML không phải là mới. Các công trình trước đó từ những năm 1990 tập trung vào mạng nơ-ron Hopfield & mạng nơ-ron tự tổ chức, độc giả quan tâm có thể tham khảo khảo sát của Smith (1999).
In another example, Larsen et al. (2018) train a neural network to predict solution of a stochastic load planning problem for which a deterministic mixed-integer linear programming formulation exists. Their main motivation: application needs to make decisions at a tactical level, i.e., under incomplete information, & ML is used to address stochasticity of problem arising from missing some of state variables in observed input. Authors use operational solutions, i.e., solutions to deterministic version of problem, & aggregate them to provide (tactical) solution targets to ML model. As explained in their paper, highest level of description of solution is its cost, whereas lowest (operational) is knowledge of values for all its variables. Then, authors place themselves in middle & predict an aggregation of variables (tactical) that corresponds to stochastic version of their specific problem. Furthermore, nature of application requires to output solutions in real time, which is not possible either for stochastic version of load planning problem or its deterministic variant when using state-of-art mixed-integer linear programming solvers. Then, ML turns out to be suitable for obtaining accurate solutions with short computing times because some of complexity is addressed offline, i.e., in learning phase, & run-time (inference) phase is extremely quick. Finally, note: in Larsen et al. (2018) an multilayer perceptron, i.e., a feedforward neural network, is used to process input instance as a vector, hence integrating very little prior knowledge about problem structure.
– Trong 1 ví dụ khác, Larsen & cộng sự (2018) đã huấn luyện 1 mạng nơ-ron để dự đoán giải pháp của 1 bài toán lập kế hoạch tải ngẫu nhiên mà tồn tại 1 công thức lập trình tuyến tính số nguyên hỗn hợp xác định. Động lực chính của họ: ứng dụng cần đưa ra quyết định ở cấp độ chiến thuật, tức là trong điều kiện thông tin không đầy đủ, & ML được sử dụng để giải quyết tính ngẫu nhiên của vấn đề phát sinh do thiếu 1 số biến trạng thái trong đầu vào quan sát được. Các tác giả sử dụng các giải pháp hoạt động, tức là các giải pháp cho phiên bản xác định của vấn đề, & tổng hợp chúng để cung cấp các mục tiêu giải pháp (chiến thuật) cho mô hình ML. Như đã giải thích trong bài báo của họ, mức độ mô tả cao nhất của giải pháp là chi phí của nó, trong khi mức độ thấp nhất (hoạt động) là kiến thức về giá trị cho tất cả các biến của nó. Sau đó, các tác giả đặt mình vào giữa & dự đoán 1 tập hợp các biến (chiến thuật) tương ứng với phiên bản ngẫu nhiên của vấn đề cụ thể của họ. Hơn nữa, bản chất của ứng dụng đòi hỏi phải đưa ra các giải pháp theo thời gian thực, điều này không thể thực hiện được đối với cả phiên bản ngẫu nhiên của bài toán lập kế hoạch tải hoặc biến thể xác định của nó khi sử dụng các bộ giải quy hoạch tuyến tính hỗn hợp số nguyên tiên tiến. Do đó, học máy (ML) tỏ ra phù hợp để thu được các giải pháp chính xác với thời gian tính toán ngắn vì 1 số độ phức tạp được xử lý ngoại tuyến, tức là trong giai đoạn học, giai đoạn chạy (suy luận) diễn ra cực kỳ nhanh. Cuối cùng, lưu ý: trong Larsen & cộng sự (2018), 1 perceptron đa lớp, tức là 1 mạng nơ-ron truyền thẳng, được sử dụng để xử lý thể hiện đầu vào dưới dạng 1 vectơ, do đó tích hợp rất ít kiến thức trước đó về cấu trúc bài toán.

∗ 3.2.2. Learning to configure algorithms. In many cases, using only ML to tackle problem may not be most suitable approach. Instead, ML can be applied to provide additional pieces of information to a combinatorial optimization algorithm as illustrated in Fig. 8: ML model is used to augment an operation research algorithm with valuable pieces of information.. E.g., ML can provide a parameterization of algorithm (in a very broad sense).
– Học cách cấu hình thuật toán. Trong nhiều trường hợp, chỉ sử dụng Học máy để giải quyết vấn đề có thể không phải là cách tiếp cận phù hợp nhất. Thay vào đó, Học máy có thể được áp dụng để cung cấp thêm thông tin cho thuật toán tối ưu hóa tổ hợp như minh họa trong Hình 8: Mô hình Học máy được sử dụng để bổ sung thông tin giá trị cho thuật toán nghiên cứu hoạt động.. Ví dụ, Học máy có thể cung cấp tham số hóa thuật toán (theo nghĩa rất rộng).
Algorithm configuration, detailed in Bischl et al. (2016); Hoos (2012), is a well studied area that captures setting presented here. Complex optimization algorithms usually have a set of parameters left constant during optimization (in ML they are called hyper-parameters). E.g., this can be aggressiveness of pre-solving operations (usually controlled by a single parameter) of an mixed-integer linear programming solver, or learning rate/step size in gradient descent methods. Carefully

selecting their value can dramatically change performance of optimization algorithm. Hence, algorithm configuration community started looking for good default parameters. Then good default parameters for different cluster of similar problem instances. From ML point of view, former is a constant regression, while 2nd is a piecewise constant nearest neighbors regression. Natural continuation was to learn a regression mapping problem instances to algorithm parameters.
– Cấu hình thuật toán, được trình bày chi tiết trong Bischl et al. (2016); Hoos (2012), là 1 lĩnh vực được nghiên cứu kỹ lưỡng, nắm bắt bối cảnh được trình bày ở đây. Các thuật toán tối ưu hóa phức tạp thường có 1 tập hợp các tham số được giữ nguyên trong quá trình tối ưu hóa (trong ML, chúng được gọi là siêu tham số). Ví dụ: điều này có thể là tính hung hăng của các hoạt động giải quyết trước (thường được kiểm soát bởi 1 tham số duy nhất) của bộ giải quy hoạch tuyến tính số nguyên hỗn hợp hoặc tốc độ học /kích thước bước trong các phương pháp giảm dần độ dốc. Việc lựa chọn cẩn thận các giá trị của chúng có thể thay đổi đáng kể hiệu suất của thuật toán tối ưu hóa. Do đó, cộng đồng cấu hình thuật toán bắt đầu tìm kiếm các tham số mặc định tốt. Sau đó, các tham số mặc định tốt cho các cụm khác nhau của các trường hợp bài toán tương tự. Theo quan điểm ML, trước đây là hồi quy hằng số, trong khi thứ 2 là hồi quy láng giềng gần nhất hằng số từng phần. Tiếp tục tự nhiên là học 1 hồi quy ánh xạ các trường hợp bài toán thành các tham số thuật toán.
In this context, Kruber, Lübbecke, & Parmentier (2017) use ML on mixed-integer linear programming instances to estimate beforehand whether or not applying a Dantzig-Wolf decomposition will be effective, i.e., will make solving time faster. Decomposition methods can be powerful but deciding if & how to apply them depends on many ingredients of instance & of its formulation & there is no clear cut way of optimally making such a decision. In their work, a data point is represented as a fixed length vector with features representing instance & tentative decomposition statistics. In another example, in context of mixed-integer quadratic programming, Bonami, Lodi, & Zarpellon (2018) use ML to decide if linearizing problem will solve faster.
– Trong bối cảnh này, Kruber, Lübbecke, & Parmentier (2017) sử dụng ML trên các trường hợp quy hoạch tuyến tính số nguyên hỗn hợp để ước tính trước liệu việc áp dụng phân tích Dantzig-Wolf có hiệu quả hay không, tức là sẽ giúp giải quyết vấn đề nhanh hơn. Các phương pháp phân tích có thể rất mạnh mẽ, nhưng việc quyết định áp dụng chúng như thế nào phụ thuộc vào nhiều yếu tố của trường hợp & công thức của nó, & không có cách rõ ràng nào để đưa ra quyết định tối ưu. Trong công trình của họ, 1 điểm dữ liệu được biểu diễn dưới dạng 1 vectơ có độ dài cố định với các đặc trưng biểu diễn trường hợp & thống kê phân tích tạm thời. Trong 1 ví dụ khác, trong bối cảnh quy hoạch bậc 2 số nguyên hỗn hợp, Bonami, Lodi, & Zarpellon (2018) sử dụng ML để quyết định xem việc tuyến tính hóa bài toán có giải quyết nhanh hơn hay không.
When quadratic programming problem given by relaxation is convex, i.e., quadratic objective matrix is semidefinite positive, one could address problem by a branch-&-bound algorithm that solves quadratic programming relaxations [Note: convex quadratic programming can be solved in polynomial time.] to provide lower bounds. Even in this convex case, not clear if quadratic programming branch-&-bound would solve faster than linearizing problem (by using McCormick (1976) inequalities) & solving an equivalent mixed-integer linear programming. This is why ML is a great candidate here to fill knowledge gap. In both papers (Bonami et al., 2018; Kruber et al., 2017), authors experiment with different ML models, e.g. support vector machines & random forests, as is good practice when no prior knowledge is embedded in model.
– Khi bài toán lập trình bậc 2 được cho bởi phép giãn ma trận là lồi, tức là ma trận mục tiêu bậc 2 là bán xác định dương, ta có thể giải quyết bài toán bằng thuật toán nhánh-&-bound để giải các phép giãn ma trận bậc 2 [Lưu ý: lập trình bậc 2 lồi có thể được giải trong thời gian đa thức.] để cung cấp các cận dưới. Ngay cả trong trường hợp lồi này, vẫn chưa rõ liệu lập trình bậc 2 nhánh-&-bound có giải quyết nhanh hơn bài toán tuyến tính hóa (bằng cách sử dụng bất đẳng thức McCormick (1976)) hay không & giải 1 bài toán lập trình tuyến tính số nguyên hỗn hợp tương đương. Đây là lý do tại sao ML là 1 ứng cử viên tuyệt vời để lấp đầy khoảng trống kiến thức. Trong cả 2 bài báo (Bonami & cộng sự, 2018; Kruber & cộng sự, 2017), các tác giả thử nghiệm với các mô hình ML khác nhau, ví dụ như máy vectơ hỗ trợ & rừng ngẫu nhiên, như 1 thông lệ tốt khi không có kiến thức trước nào được nhúng vào mô hình.
Heuristic building framework used in Karapetyan et al. (2017) & Mascia et al. (2014), already presented in Sect. 3.1.2, can be understood under this eye. Indeed, it can be seen as a large parametric heuristic, configured by transition probabilities in former case, & by parameter representing a sentence in the latter.
– Khung xây dựng heuristic được sử dụng trong Karapetyan & cộng sự (2017) & Mascia & cộng sự (2014), đã được trình bày trong Phần 3.1.2, có thể được hiểu theo góc nhìn này. Thật vậy, nó có thể được xem như 1 heuristic tham số lớn, được cấu hình bởi xác suất chuyển tiếp trong trường hợp trước, & bởi tham số biểu diễn 1 câu trong trường hợp sau.
As previously stated, parameterization of combinatorial optimization algorithm provided by ML is to be understood in a very broad sense. E.g., in case of radiation therapy for cancer treatment, Mahmood, Babier, McNiven, Diamant, & Chan (2018) use ML to produce candidate therapies that are afterward refined by a combinatorial optimization algorithm into a deliverable plan. Namely, a generative adversarial network is used to color CT scan images into a potential radiation plan, then, inverse optimization (Ahuja & Orlin, 2001) is applied on result to make plan feasible (Chan, Craig, Lee, & Sharpe, 2014). In general, generative adversarial network are made of 2 distinct networks: one (generator) generates images, & another one (discriminator) discriminates between generated images & a dataset of real images. Both are trained alternatively: discriminator through a usual supervised objective, while generator is trained to fool discriminator. In Mahmood et al. (2018), a particular type of generative adversarial network (conditional generative adversarial network) is used to provide coloring instead of random images. Interested reader is referred to Creswell et al. (2018) for an overview on generative adversarial network.
– Như đã nêu trước đó, việc tham số hóa thuật toán tối ưu hóa tổ hợp do ML cung cấp cần được hiểu theo nghĩa rất rộng. Ví dụ, trong trường hợp xạ trị để điều trị ung thư, Mahmood, Babier, McNiven, Diamant, & Chan (2018) sử dụng ML để tạo ra các liệu pháp ứng viên, sau đó được tinh chỉnh bằng thuật toán tối ưu hóa tổ hợp thành 1 kế hoạch có thể

phân phối. Cụ thể, 1 mạng đối nghịch sinh sản được sử dụng để tô màu hình ảnh chụp CT thành 1 kế hoạch xạ trị tiềm năng, sau đó, tối ưu hóa ngược (Ahuja & Orlin, 2001) được áp dụng trên kết quả để làm cho kế hoạch khả thi (Chan, Craig, Lee, & Sharpe, 2014). Nhìn chung, mạng đối nghịch sinh sản được tạo thành từ 2 mạng riêng biệt: 1 (bộ tạo) tạo ra hình ảnh, & 1 mạng khác (bộ phân biệt) phân biệt giữa các hình ảnh được tạo & 1 tập dữ liệu hình ảnh thực. Cả 2 được đào tạo luân phiên: bộ phân biệt thông qua 1 mục tiêu được giám sát thông thường, trong khi bộ tạo được đào tạo để đánh lừa bộ phân biệt. Trong Mahmood et al. (2018), 1 loại mạng đối kháng sinh sinh cụ thể (mạng đối kháng sinh sinh có điều kiện) được sử dụng để tô màu thay cho hình ảnh ngẫu nhiên. Độc giả quan tâm có thể tham khảo Creswell & cộng sự (2018) để biết tổng quan về mạng đối kháng sinh sinh.

End this sect by noting:: an ML model used for learning some representation may in turn use as features pieces of information given by another combinatorial optimization algorithm, e.g. decomposition statistics used in Kruber et al. (2017), or the linear programming information in Bonami et al. (2018). Moreover, remark: in satisfiability context, learning of type of algorithm to execute on a particular cluster of instances has been paired with learning of parameters of the algorithm itself, see, e.g., Ansótegui, Heymann, Pon, Sellmann, & Tierney (2019); Ansótegui, Pon, Sellmann, & Tierney (2017).

– Kết thúc phần này bằng cách lưu ý: 1 mô hình ML được sử dụng để học 1 số biểu diễn có thể sử dụng các thông tin được cung cấp bởi 1 thuật toán tối ưu hóa tổ hợp khác, ví dụ: thống kê phân rã được sử dụng trong Kruber & cộng sự (2017), hoặc thông tin lập trình tuyến tính trong Bonami & cộng sự (2018), làm đặc trưng. Hơn nữa, lưu ý: trong bối cảnh khả năng thỏa mãn, việc học loại thuật toán để thực thi trên 1 cụm trường hợp cụ thể đã được kết hợp với việc học các tham số của chính thuật toán đó, xem ví dụ: Ansótegui, Heymann, Pon, Sellmann & Tierney (2019); Ansótegui, Pon, Sellmann, & Tierney (2017).

∗ 3.2.3. ML alongside optimization algorithms. To generalize context of previous sect to its full potential, one can build combinatorial optimization algorithms that repeatedly call an ML model throughout their execution, as illustrated in Fig. 9: Combinatorial optimization algorithm repeatedly requires same ML model to make decisions. ML model takes as input current state of algorithm, which may include problem definition. A master algorithm controls high-level structure while frequently calling an ML model to assist in lower level decisions. Key difference between this approach & examples discussed in previous sect: *same ML model* is used by combinatorial optimization algorithm to make same type of decisions a number of times in order of number of iterations of algorithm. As in prev sect, nothing prevents one from applying additional steps before or after such an algorithm.

– ML cùng với các thuật toán tối ưu hóa. Để khái quát hóa toàn bộ bối cảnh của phần trước, người ta có thể xây dựng các thuật toán tối ưu hóa tổ hợp liên tục gọi 1 mô hình ML trong suốt quá trình thực thi của chúng, như minh họa trong Hình 9: Thuật toán tối ưu hóa tổ hợp yêu cầu nhiều lần cùng 1 mô hình ML để đưa ra quyết định. Mô hình ML lấy trạng thái hiện tại của thuật toán làm đầu vào, có thể bao gồm định nghĩa vấn đề. Thuật toán chính kiểm soát cấu trúc cấp cao trong khi thường xuyên gọi 1 mô hình ML để hỗ trợ các quyết định cấp thấp hơn. Sự khác biệt chính giữa cách tiếp cận này & các ví dụ được thảo luận trong phần trước: *cùng 1 mô hình ML* được thuật toán tối ưu hóa tổ hợp sử dụng để đưa ra cùng 1 loại quyết định nhiều lần theo thứ tự số lần lặp của thuật toán. Như trong phần trước, không có gì ngăn cản người ta áp dụng các bước bổ sung trước hoặc sau 1 thuật toán như vậy.

This is clearly context of branch-&-bound tree for mixed-integer linear programming, where already mentioned how task of selecting branching variable is either too heuristic or too slow, & is therefore a good candidate for learning (Lodi & Zarpellon, 2017). In this case, general algorithm remains a branch-&-bound framework, with same software architecture & same guarantees on lower & upper bounds, but branching decisions made at every node are left to be learned. Likewise, work of Hottung et al. (2017) learning both a branching policy & value network for heuristic tree search undeniably fits in this context. Another important aspect in solving mixed-integer linear programming is use of primal heuristics, i.e., algorithms that are applied in branch-&-bound nodes to find feasible solutions, without guarantee of success. On top of their obvious advantages, good solutions also give tighter upper bounds (for minimization problems) on solution value & make more pruning of tree possible. Heuristics depend on branching node (as branching fix some variables to specific values), so they need to be run frequently. However, running them too often can slow down exploration of tree, especially if their outcome is negative, i.e., no better upper bound is detected. Khalil, Dilkina, Nemhauser, Ahmed, & Shao (2017b) build a ML model to predict whether or not running a given heuristic will yield a better solution than best one found so far & then greedily run that heuristic whenever outcome of model is positive.

– Đây rõ ràng là ngữ cảnh của cây nhánh-&-bound cho lập trình tuyến tính số nguyên hỗn hợp, trong đó đã đề cập đến cách nhiệm vụ chọn biến nhánh quá kinh nghiệm hoặc quá chậm, do đó là 1 ứng cử viên tốt cho việc học (Lodi & Zarpellon, 2017). Trong trường hợp này, thuật toán tổng quát vẫn là 1 khuôn khổ nhánh-&-bound, với cùng kiến trúc phần mềm & cùng đảm bảo về các giới hạn dưới & trên, nhưng các quyết định phân nhánh được đưa ra tại mỗi nút vẫn phải học. Tương tự như vậy, công trình của Hottung & cộng sự (2017) về việc học cả chính sách phân nhánh & mạng giá trị cho tìm kiếm cây kinh nghiệm chắc chắn phù hợp với ngữ cảnh này. Một khía cạnh quan trọng khác trong việc giải quyết lập trình tuyến tính số nguyên hỗn hợp là việc sử dụng các kinh nghiệm nguyên thủy, tức là các thuật toán được áp dụng trong các nút nhánh-&-bound để tìm các giải pháp khả thi, mà không đảm bảo thành công. Bên cạnh những lợi thế rõ ràng của chúng, các giải pháp tốt còn cung cấp các giới hạn trên chặt chẽ hơn (cho các vấn đề tối thiểu hóa) về giá trị giải pháp & giúp có thể cắt tỉa cây nhiều hơn. Thuật toán heuristic phụ thuộc vào nút phân nhánh (vì phân nhánh cố định 1 số biến thành các giá trị cụ thể), do đó chúng cần được chạy thường xuyên. Tuy nhiên, việc chạy chúng quá thường xuyên có thể làm chậm quá trình khám phá cây, đặc biệt nếu kết quả của chúng là âm, tức là không tìm thấy giới hạn trên tốt hơn. Khalil, Dilkina, Nemhauser, Ahmed, & Shao (2017b) xây dựng 1 mô hình ML để dự đoán liệu việc chạy 1 thuật toán heuristic nhất định có mang lại giải pháp tốt hơn giải pháp tốt nhất đã tìm thấy cho đến nay hay không &

sau đó chạy thuật toán heuristic đó 1 cách tham lam bất cứ khi nào kết quả của mô hình là dương.
Approximation used by Baltean-Lugojan et al. (2018), already discussed in Sect. 3.2.1, is an example of predicting a high-level description of solution to an optimization problem, namely objective value. Nonetheless, goal: solve original quadratic programming. Thus, learned model is queried repeatedly to select promising cutting planes. ML model is used only to select promising cuts, but once selected, cuts are added to linear programming relaxation, thus embedding ML outcome into an exact algorithm. This approach highlights promising directions for this type of algorithm. Decision learned is critical because adding best cutting planes is necessary for solving problem fast (or fast enough, because in presence of NP-hard problems, optimization may time out before any meaningful solving). At same time, approximate decision (often in form of a probability) does not compromise exactness of algorithm: any cut added is guaranteed to be valid. This setting leaves room for ML to thrive, while reducing need for guarantees from ML algorithms (an active & difficult area of research). In addition, note: approach in Larsen et al. (2018) is part of a master algorithm in which ML is iteratively invoked to make booking decisions in real time. Work of Khalil et al. (2017a), presented in Sect. 3.1.2, also belongs to this setting, even if resulting algorithm is heuristic. Indeed, an ML model is asked to select most relevant node, while a master algorithm maintains partial tour, computes its length, etc. Because master algorithm is very simple, possible to see contribution as an end-to-end method, as stated in Sect. 3.2.1, but it can also be interpreted more generally as done here.
– Xấp xỉ được Baltean-Lugojan & cộng sự (2018) sử dụng, đã được thảo luận trong Phần 3.2.1, là 1 ví dụ về việc dự đoán mô tả cấp cao về giải pháp cho 1 bài toán tối ưu hóa, cụ thể là giá trị khách quan. Tuy nhiên, mục tiêu: giải quyết bài toán quy hoạch bậc 2 ban đầu. Do đó, mô hình đã học được truy vấn nhiều lần để chọn các mặt cắt hứa hẹn. Mô hình ML chỉ được sử dụng để chọn các mặt cắt hứa hẹn, nhưng 1 khi đã được chọn, các mặt cắt sẽ được thêm vào phần nới lỏng quy hoạch tuyến tính, do đó nhúng kết quả ML vào 1 thuật toán chính xác. Cách tiếp cận này làm nổi bật các hướng đầy hứa hẹn cho loại thuật toán này. Quyết định đã học được là rất quan trọng vì việc thêm các mặt cắt tốt nhất là cần thiết để giải quyết vấn đề nhanh chóng (hoặc đủ nhanh, vì khi có các bài toán NP-khó, quá trình tối ưu hóa có thể hết thời gian trước khi có bất kỳ lời giải có ý nghĩa nào). Đồng thời, quyết định xấp xỉ (thường ở dạng xác suất) không làm giảm độ chính xác của thuật toán: bất kỳ mặt cắt nào được thêm vào đều được đảm bảo là hợp lệ. Thiết lập này tạo điều kiện cho ML phát triển mạnh, đồng thời giảm nhu cầu đảm bảo từ các thuật toán ML (một lĩnh vực nghiên cứu đang được quan tâm & khó khăn). Ngoài ra, lưu ý: cách tiếp cận trong Larsen & cộng sự (2018) là 1 phần của thuật toán chính, trong đó ML được gọi lặp đi lặp lại để đưa ra quyết định đặt chỗ theo thời gian thực. Nghiên cứu của Khalil & cộng sự (2017a), được trình bày trong Phần 3.1.2, cũng thuộc trường hợp này, ngay cả khi thuật toán kết quả mang tính chất heuristic. Thật vậy, 1 mô hình ML được yêu cầu chọn nút có liên quan nhất, trong khi thuật toán chính duy trì 1 phần hành trình, tính toán độ dài của nó, v.v. Vì thuật toán chính rất đơn giản, có thể xem đóng góp như 1 phương pháp đầu cuối, như đã nêu trong Phần 3.2.1, nhưng cũng có thể được hiểu 1 cách tổng quát hơn là được thực hiện ở đây.
Markov Chain framework for building heuristics from Karapetyan et al. (2017) can also be framed as repeated decisions. Transition matrix can be queried & sampled from in order to transition from 1 state to another, i.e., to make low level decisions of choosing next move. 3 distinctions made in this Sect. 3.2 are general enough that they can overlap. Here, fact that model operates on internal state transitions, yet is learned globally, is what makes it hard to analyze.
– Khung chuỗi Markov để xây dựng các phương pháp heuristic từ Karapetyan & cộng sự (2017) cũng có thể được định hình dưới dạng các quyết định lặp lại. Ma trận chuyển đổi có thể được truy vấn & lấy mẫu để chuyển từ trạng thái này sang trạng thái khác, tức là để đưa ra các quyết định cấp thấp về việc chọn bước tiếp theo. 3 điểm khác biệt được nêu trong Mục 3.2 này khá tổng quát đến mức chúng có thể chồng chéo lên nhau. Ở đây, thực tế là mô hình hoạt động dựa trên các chuyển đổi trạng thái nội bộ, nhưng lại được học toàn cục, chính là lý do khiến việc phân tích trở nên khó khăn.
Before ending this sect, worth mentioning: learning recurrent algorithmic decisions is also used in DL community, e.g., in field of meta-learning to decide how to apply gradient updates in stochastic gradient descent (Andrychowicz et al., 2016; Li & Malik, 2017; Wichrowska et al., 2017).
– Trước khi kết thúc phần này, điều đáng đề cập là: việc học các quyết định thuật toán tuần hoàn cũng được sử dụng trong cộng đồng DL, ví dụ, trong lĩnh vực siêu học để quyết định cách áp dụng các bản cập nhật gradient trong quá trình giảm dần gradient ngẫu nhiên (Andrychowicz & cộng sự, 2016; Li & Malik, 2017; Wichrowska & cộng sự, 2017).

- 4. Learning objective. In prev sect, have surveyed existing literature by orthogonally grouping main contributions of ML for combinatorial optimization into families of approaches, sometimes with overlaps. In this sect, formulate & study objective that drives learning process.

  – Mục tiêu học tập. Trong phần trước, chúng tôi đã khảo sát các tài liệu hiện có bằng cách nhóm trực giao các đóng góp chính của ML cho tối ưu hóa tổ hợp thành các nhóm phương pháp tiếp cận, đôi khi có sự chồng chéo. Trong phần này, chúng tôi xây dựng & mục tiêu nghiên cứu thúc đẩy quá trình học tập.

  - 4.1. Multi-instance formulation. In following, introduce an abstract learning formulation (inspired from Bischl et al. (2016)). How would an ML practitioner compare optimization algorithms? Define $\mathcal{I}$ to be a set of problem instances, & P a probability distribution over $\mathcal{I}$. These are problems that we care about, weighted by a probability distribution, reflecting fact: in a practical application, not all problems are as likely. In practice, $\mathcal{I}$ or P are inaccessible, but can observe some samples from P, as motivated in introduction with Montreal delivery company. For a set of algorithms $\mathcal{A}$, let $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ be a measure of performance of an algorithm on a problem instance (lower is better). This could be objective value of best solution found, but could also incorporate elements from optimality bounds, absence of results, running times, & resource

usage. To compare $a_1, a_2 \in \mathcal{A}$, an ML practitioner would compare $\mathbb{E}_{i\sim P} m(i, a_1), \mathbb{E}_{i\sim P} m(i, a_2)$, or equivalently (4)

$$\min_{a\in\{a_1,a_2\}} \mathbb{E}_{i\sim P} m(i, a).$$

Because measuring these quantities is not tractable, one will typically use empirical estimates instead, by using a finite dataset D_{train} of independent instances sampled from P: (5)

$$\min_{a\in\{a_1,a_2\}} \sum_{i\in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} m(i, a).$$

This is intuitive & done in practice: collect a dataset of problem instances & compare say, average running times. Of course, such expectation can be computed for different datasets (different $\mathcal{I}$'s & P's), & different measures (different m's).

– Công thức đa trường hợp. Sau đây, hãy giới thiệu 1 công thức học trừu tượng (lấy cảm hứng từ Bischl & cộng sự (2016)). Một người thực hành ML sẽ so sánh các thuật toán tối ưu hóa như thế nào? Định nghĩa $\mathcal{I}$ là 1 tập hợp các trường hợp vấn đề, & P là phân phối xác suất trên $\mathcal{I}$. Đây là những vấn đề mà chúng ta quan tâm, được cân nhắc theo phân phối xác suất, phản ánh thực tế: trong ứng dụng thực tế, không phải tất cả các vấn đề đều có khả năng xảy ra như nhau. Trong thực tế, $\mathcal{I}$ hoặc P không thể truy cập được, nhưng có thể quan sát 1 số mẫu từ P, như đã nêu trong phần giới thiệu với công ty giao hàng Montreal. Đối với 1 tập hợp các thuật toán $\mathcal{A}$, hãy để $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ là thước đo hiệu suất của 1 thuật toán trên 1 trường hợp vấn đề (càng thấp càng tốt). Đây có thể là giá trị khách quan của giải pháp tốt nhất được tìm thấy, nhưng cũng có thể kết hợp các yếu tố từ ranh giới tối ưu, không có kết quả, thời gian chạy, & mức sử dụng tài nguyên. Để so sánh $a_1, a_2 \in \mathcal{A}$, 1 người thực hành ML sẽ so sánh $\mathbb{E}_{i\sim P} m(i, a_1), \mathbb{E}_{i\sim P} m(i, a_2)$, hoặc tương đương (4)

$$\min_{a\in\{a_1,a_2\}} \mathbb{E}_{i\sim P} m(i, a).$$

Vì việc đo lường các đại lượng này không dễ dàng, nên người ta thường sử dụng các ước tính thực nghiệm thay thế, bằng cách sử dụng 1 tập dữ liệu hữu hạn D_{train} gồm các trường hợp độc lập được lấy mẫu từ P: (5)

$$\min_{a\in\{a_1,a_2\}} \sum_{i\in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} m(i, a).$$

Điều này rất trực quan & được thực hiện trong thực tế: thu thập 1 tập dữ liệu các trường hợp bài toán & so sánh, chẳng hạn, thời gian chạy trung bình. Tất nhiên, kỳ vọng như vậy có thể được tính toán cho các tập dữ liệu khác nhau (các $\mathcal{I}$ khác nhau & P khác nhau), & các phép đo khác nhau (các m khác nhau).

This is already a learning problem. More general one that we want to solve through learning is: (6)

$$\min_{a\in\mathcal{A}} \mathbb{E}_{i\sim P} m(i, a).$$

Instead of comparing between 2 algorithms, may compare among an uncountable, maybe non-parametric, space of algorithms. To see how we com up with so many algorithms, have to look at algorithms in Sect. 3, & think of ML model space over which we learn as defining parametrizing algorithm space $\mathcal{A}$. E.g., consider case of learning a branching policy π for branch-&-bound. If define policy to be a neural network with a set of weights $\theta \in \mathbb{R}^p$, then obtain a parametric branch-&-bound algorithm $a(\pi_\theta)$ & (6) becomes (7)

$$\min_{\theta\in\mathbb{R}^p} \mathbb{E}_{i\sim P} m(i, a(\pi_\theta)).$$

– Đây vốn đã là 1 bài toán học tập. Một bài toán tổng quát hơn mà chúng ta muốn giải quyết thông qua học tập là: (6)

$$\min_{a\in\mathcal{A}} \mathbb{E}_{i\sim P} m(i, a).$$

Thay vì so sánh giữa 2 thuật toán, chúng ta có thể so sánh giữa 1 không gian thuật toán không đếm được, có thể là phi tham số. Để thấy cách chúng ta tạo ra nhiều thuật toán như vậy, hãy xem xét các thuật toán trong Phần 3, & nghĩ về không gian mô hình ML mà chúng ta học tập như việc xác định không gian thuật toán tham số hóa $\mathcal{A}$. Ví dụ, hãy xem xét trường hợp học 1 chính sách phân nhánh π cho cận nhánh. Nếu định nghĩa chính sách là 1 mạng nơ-ron với 1 tập hợp các trọng số $\theta \in \mathbb{R}^p$, thì thu được 1 thuật toán nhánh tham số $a(\pi_\theta)$ & (6) trở thành (7)

$$\min_{\theta\in\mathbb{R}^p} \mathbb{E}_{i\sim P} m(i, a(\pi_\theta)).$$

Unfortunately, solving this problem is hard. On 1 hand, performance measure m is most often not differentiable & without closed form expression. Discuss this in Sect. 4.2. On other hand, computing expectation in (6) is intractable. As in (5), one can use an empirical distribution using a finite dataset, but that leads to *generalization* considerations, as explained in Sect. 4.3.

– Thật không may, việc giải quyết vấn đề này rất khó khăn. Một mặt, thước đo hiệu suất m thường không khả vi & nếu không có biểu thức dạng đóng. Thảo luận về điều này trong Mục 4.2. Mặt khác, việc tính toán kỳ vọng trong (6) rất khó khăn. Cũng như trong (5), ta có thể sử dụng phân phối thực nghiệm bằng cách sử dụng 1 tập dữ liệu hữu hạn, nhưng điều đó dẫn đến những cân nhắc về *tổng quát hóa*, như đã giải thích trong Mục 4.3.

Before move on, introduce a new element to make (6) more general. That formula suggests: once given an instance, outcome of performance measure is deterministic. That is unrealistic for multiple reasons. Performance measure could itself incorporate some source of randomness due to external factors, e.g. with running times which are hardware & system dependent. Algorithm could also incorporate non negligible sources of randomness, if it is designed to be stochastic, or if some operations are non deterministic, or to express fact: algorithm should be robust to choice of some external parameters. Let τ be that source of randomness, $\pi \in \Pi$ interval policy being learned, & $a(\pi, \tau)$ resulting algorithm, then can reformulate (6) as (8)

$$\min_{\pi \in \Pi} \mathbb{E}_{i \sim P}[\mathbb{E}_{\tau}[m(i, a(\pi, \tau))|i]].$$

In particular, when learning repeated decisions, as in Sect. 3.2.3, this source of randomness can be expressed along trajectory followed in Markov decision process, using dynamics of environment $p(s', r|a, s)$ (see Fig. 2). Addition made in (8) will be useful for discussion on generalization in Sect. 4.3.

– Trước khi tiếp tục, hãy giới thiệu 1 phần tử mới để làm cho (6) tổng quát hơn. Công thức đó gợi ý: khi cho 1 trường hợp, kết quả của phép đo hiệu suất là xác định. Điều này không thực tế vì nhiều lý do. Bản thân phép đo hiệu suất có thể kết hợp 1 số nguồn ngẫu nhiên do các yếu tố bên ngoài, ví dụ: với thời gian chạy phụ thuộc vào phần cứng & hệ thống. Thuật toán cũng có thể kết hợp các nguồn ngẫu nhiên không đáng kể, nếu nó được thiết kế để ngẫu nhiên, hoặc nếu 1 số phép toán không xác định, hoặc để biểu diễn thực tế: thuật toán phải mạnh mẽ đối với việc lựa chọn 1 số tham số bên ngoài. Giả sử τ là nguồn ngẫu nhiên đó, chính sách khoảng $\pi \in \Pi$ đang được học, & $a(\pi, \tau)$ thuật toán kết quả, sau đó có thể xây dựng lại (6) thành (8)

$$\min_{\pi \in \Pi} \mathbb{E}_{i \sim P}[\mathbb{E}_{\tau}[m(i, a(\pi, \tau))|i]].$$

Đặc biệt, khi học các quyết định lặp lại, như trong Mục 3.2.3, nguồn ngẫu nhiên này có thể được biểu diễn dọc theo quỹ đạo được theo sau trong quy trình quyết định Markov, sử dụng động lực học của môi trường $p(s', r|a, s)$ (xem Hình 2). Phần bổ sung trong (8) sẽ hữu ích cho việc thảo luận về khái quát hóa trong Mục 4.3.

- 4.2. Surrogate objectives. In prev sect, have formulated a proper learning objective. Here, try to relate that objective to learning methods of Sect. 3.1, namely, demonstration & experience. If usual learning metrics of an ML model, e.g., accuracy for classification in supervised (imitation) learning, is improving, does it mean: performance metric of (6) is also improving?

  – Mục tiêu thay thế. Trong phần trước, đã xây dựng 1 mục tiêu học tập phù hợp. Ở đây, hãy thử liên hệ mục tiêu đó với các phương pháp học tập của Phần 3.1, cụ thể là minh họa & trải nghiệm. Nếu các số liệu học tập thông thường của 1 mô hình ML, ví dụ như độ chính xác phân loại trong học có giám sát (bắt chước), đang được cải thiện, liệu điều đó có nghĩa là: số liệu hiệu suất của (6) cũng đang được cải thiện không?

  A straightforward approach for solving (8) is that of RL (including direct optimization methods), as surveyed in Sect. 3.1.2. Objective from (6) can be optimized directly on experience data by matching total return to performance measure. Sometimes, a single final reward can naturally be decoupled across trajectory. E.g., if performance objective of a branch-&-bound variable selection policy: minimize number of opened nodes, then policy can receive a reward discouraging an increase in number of nodes, hence giving an incentive to select variables that lead to pruning. However, that may not be always possible, leaving only option of delaying a single reward to end of trajectory. This sparse reward setting is challenging for RL algorithms, & one might want to design a surrogate reward signal to encourage intermediate accomplishments. This introduces some discrepancies, & policy being optimized may learn a behavior not intended by algorithm designer. There is a priori no relationship between 2 reward signals. One needs to make use of their intuition to design surrogate signals, e.g., minimizing number of branch-&-bound nodes *should* lead to smaller running times. Reward shaping is an active area of research in RL, yet it is often performed by a number of engineering tricks.

  – Một cách tiếp cận đơn giản để giải (8) là RL (bao gồm các phương pháp tối ưu hóa trực tiếp), như đã khảo sát trong Phần 3.1.2. Mục tiêu từ (6) có thể được tối ưu hóa trực tiếp trên dữ liệu kinh nghiệm bằng cách khớp tổng lợi nhuận với thước đo hiệu suất. Đôi khi, 1 phần thưởng cuối cùng duy nhất có thể được tách rời 1 cách tự nhiên trên toàn bộ quỹ đạo. Ví dụ: nếu mục tiêu hiệu suất của chính sách lựa chọn biến ràng buộc nhánh: giảm thiểu số lượng nút đã mở, thì chính sách có thể nhận được phần thưởng ngăn cản việc tăng số lượng nút, do đó tạo động lực để lựa chọn các biến dẫn đến việc cắt tỉa. Tuy nhiên, điều đó không phải lúc nào cũng khả thi, chỉ còn lại lựa chọn trì hoãn 1 phần thưởng duy nhất đến cuối quỹ đạo. Thiết lập phần thưởng thưa thớt này là 1 thách thức đối với các thuật toán RL, & người ta có thể muốn thiết kế 1 tín hiệu phần thưởng thay thế để khuyến khích các thành tích trung gian. Điều này dẫn đến 1 số điểm khác biệt, & chính sách được tối ưu hóa có thể học được 1 hành vi không theo ý định của nhà thiết kế thuật toán. Không có mối quan hệ nào giữa 2 tín hiệu phần thưởng. Người ta cần vận dụng trực giác của mình để thiết kế các tín hiệu thay thế, ví dụ, giảm thiểu số lượng nút liên kết nhánh điều này sẽ dẫn đến thời gian chạy ngắn hơn. Định hình phần thưởng là 1 lĩnh vực nghiên cứu đang được quan tâm trong RL, nhưng nó thường được thực hiện bằng 1 số thủ thuật kỹ thuật.

  In case of learning a policy from a supervised signal from expert demonstration, performance measure m does not even appear in learning problem that is solved. In this context, goal: optimize a policy $\pi \in \Pi$ in action space to mimic an expert policy π_e (as 1st introduced with Fig. 5)

$$\min_{\pi \in \Pi} \mathbb{E}_{i \sim P}[\mathbb{E}_{s}[l(\pi(s), \pi_e(s))|i, \pi_e]],$$

where l is a task dependent loss (classification, regression, etc.). Have emphasized that state S is conditional, not only on instance, but also on expert policy π_e used to collect data. Intuitively, better ML model learns, i.e., better policy imitates expert, closer final performance of learned policy should be to performance of expert. Under some conditions, possible to relate performance of learned policy to performance of expert policy, but covering this aspect is out of scope of this paper.

Opposite is not true, if learning fails, policy may still turn out to perform well (by encountering an alternative good decision). Indeed, when making a decision with high surrogate objective error, learning will be fully penalized when, in fact, decision could have good performances by original metric. For that reason, it is capital to report performance metrics. E.g., surveyed in Sect. 3.2.2 work of Bonami et al. (2018) where authors train a classifier to predict if a mixed integer quadratic problem instance should be linearized or not. Targets used for learner are computed optimally by solving problem instance in both configurations. Simply reporting classification accuracy is not enough. Indeed, this metric gives no information on impact a misclassification has on running times, metric used to compute targets. In binary classification, a properly classified example could also happen to have unsignificant difference between running times of 2 configurations. To alleviate this issue, authors also introduce a category where running times are not significatively different (& report real running times). A continuous extension would be to learn a regression of solving time. However, learning this regression now means: final algorithm needs to optimize over set of decisions to find best one. In RL, this is analogous to learning a value function (see Sect. 2.2). Applying same reasoning to repeated decisions is better understood with complete RL theory.

– Trong trường hợp học 1 chính sách từ tín hiệu có giám sát từ trình diễn của chuyên gia, thước đo hiệu suất m thậm chí không xuất hiện trong bài toán học đã được giải. Trong bối cảnh này, mục tiêu: tối ưu hóa 1 chính sách $\pi \in \Pi$ trong không gian hành động để mô phỏng chính sách của chuyên gia π_e (như đã giới thiệu lần đầu với Hình 5)

$$\min_{\pi \in \Pi} \mathbb{E}_{i \sim P}[\mathbb{E}_s[l(\pi(s), \pi_e(s))|i, \pi_e]],$$

trong đó l là tổn thất phụ thuộc vào tác vụ (phân loại, hồi quy, v.v.). Chúng tôi đã nhấn mạnh rằng trạng thái S có điều kiện, không chỉ phụ thuộc vào trường hợp, mà còn phụ thuộc vào chính sách của chuyên gia π_e được sử dụng để thu thập dữ liệu. Về mặt trực quan, mô hình học máy tốt hơn sẽ học, tức là chính sách tốt hơn sẽ mô phỏng chính sách của chuyên gia, hiệu suất cuối cùng của chính sách đã học sẽ gần với hiệu suất của chuyên gia hơn. Trong 1 số điều kiện, có thể liên hệ hiệu suất của chính sách đã học với hiệu suất của chính sách chuyên gia, nhưng việc đề cập đến khía cạnh này nằm ngoài phạm vi của bài báo này. Điều ngược lại là không đúng, nếu việc học không thành công, chính sách vẫn có thể hoạt động tốt (bằng cách gặp phải 1 quyết định tốt thay thế). Thật vậy, khi đưa ra quyết định với sai số mục tiêu thay thế cao, việc học sẽ bị phạt hoàn toàn trong khi trên thực tế, quyết định có thể có hiệu suất tốt theo số liệu ban đầu. Vì lý do đó, việc báo cáo các số liệu hiệu suất là rất quan trọng. Ví dụ, được khảo sát trong Phần 3.2.2, công trình của Bonami & cộng sự (2018), trong đó các tác giả huấn luyện 1 bộ phân loại để dự đoán liệu 1 trường hợp bài toán bậc 2 nguyên hỗn hợp có nên được tuyến tính hóa hay không. Các mục tiêu được sử dụng cho bộ học được tính toán tối ưu bằng cách giải quyết trường hợp bài toán trong cả 2 cấu hình. Chỉ báo cáo độ chính xác phân loại là không đủ. Thật vậy, số liệu này không cung cấp thông tin về tác động của việc phân loại sai lên thời gian chạy, số liệu được sử dụng để tính toán các mục tiêu. Trong phân loại nhị phân, 1 ví dụ được phân loại đúng cũng có thể có sự khác biệt không đáng kể giữa thời gian chạy của 2 cấu hình. Để giảm thiểu vấn đề này, các tác giả cũng giới thiệu 1 phạm trù trong đó thời gian chạy không khác biệt đáng kể (& báo cáo thời gian chạy thực tế). Một cách mở rộng liên tục sẽ là học hồi quy thời gian giải. Tuy nhiên, việc học hồi quy này giờ đây có nghĩa là: thuật toán cuối cùng cần tối ưu hóa trên 1 tập hợp các quyết định để tìm ra quyết định tốt nhất. Trong RL, điều này tương tự như việc học 1 hàm giá trị (xem Mục 2.2). Việc áp dụng cùng 1 lập luận cho các quyết định lặp lại sẽ được hiểu rõ hơn với lý thuyết RL hoàn chỉnh.

- 4.3. On generalization. In Sect. 4.1, have claimed: probability distribution in (6) is inaccessible & needs to be replaced by empirical probability distribution over a finite dataset D_{train}. Optimization problem solved is (10)

$$\min_{a \in \mathcal{A}} \sum_{i \in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} m(i, a).$$

As pointed out in Sect. 2,2, when optimizing over empirical probability distribution, risk having a low performance measure on finite number of problem instances, *regardless of true probability distribution*. In this case, *generalization* error is high because of discrepancy between training performances & true expected performances (overfitting). To control this aspect, a validation set D_{valid} is introduced to compare a finite number of candidate algorithms based on estimates of generalization performances, & a test set D_{test} is used for estimating generalization performances of selected algorithm.

– Về tổng quát hóa. Trong Mục 4.1, đã khẳng định: phân phối xác suất trong (6) không thể tiếp cận được & cần được thay thế bằng phân phối xác suất thực nghiệm trên 1 tập dữ liệu hữu hạn D_{train}. Bài toán tối ưu hóa được giải quyết là (10)

$$\min_{a \in \mathcal{A}} \sum_{i \in D_{\text{train}}} \frac{1}{|D_{\text{train}}|} m(i, a).$$

Như đã chỉ ra trong Mục 2,2, khi tối ưu hóa trên phân phối xác suất thực nghiệm, có nguy cơ có thước đo hiệu suất thấp trên 1 số lượng hữu hạn các trường hợp bài toán, *bất kể phân phối xác suất thực*. Trong trường hợp này, lỗi tổng quát hóa cao do sự khác biệt giữa hiệu suất huấn luyện & hiệu suất kỳ vọng thực (quá khớp). Để kiểm soát khía cạnh này, 1 tập hợp xác thực D_{valid} được đưa vào để so sánh 1 số lượng hữu hạn các thuật toán ứng viên dựa trên ước tính hiệu suất tổng quát hóa, & 1 tập hợp thử nghiệm D_{test} được sử dụng để ước tính hiệu suất tổng quát hóa của thuật toán đã chọn.

In following, look more intuitively at generalization in ML for combinatorial optimization, & its consequences. To make it easier, recall different learning scenarios. In introduction, have motivated Montreal delivery company example, where problems of interest are from an unknown probability distribution of Montreal TSP. This is a very restricted set of problems, but enough to deliver value for this business. Much more ambitious, may want our policy learned on a finite set of instances

to perform well (generalize) to any "real-world" mixed-integer linear programming instance. This is of interest if you are in business of selling mixed-integer linear programming solvers & want branching policy to perform well for as many of your clients as possible. In both cases, generalization applies to instances that are not known to algorithm implementer. These are only instances that we care about; the one used for training are already solved. Topic of probability distribution of instances also appears naturally in stochastic programming/optimization, where uncertainty about problem is modeled through probability distributions. Scenario generation, an essential way to solve this type of optimization programs, require sampling from this distribution & solving associated problem multiple times. Nair, Dvijotham, Dunning, & Vinyals (2018) take advantage of this repetitive process to learn an end-to-end model to solve problem. Their model is composed of a local search & a local improvement policy & is trained through RL. Here, generalization means: during scenario generalization, learned search beats other approaches, hence delivering an overall faster stochastic programming algorithm. In short, *learning without generalization is pointless!*

– Tiếp theo, hãy xem xét 1 cách trực quan hơn về khái quát hóa trong ML đối với tối ưu hóa tổ hợp, & các hệ quả của nó. Để dễ hiểu hơn, hãy nhớ lại các kịch bản học tập khác nhau. Trong phần giới thiệu, chúng tôi đã lấy ví dụ về công ty giao hàng Montreal, trong đó các vấn đề quan tâm xuất phát từ 1 phân phối xác suất chưa biết của Montreal TSP. Đây là 1 tập hợp các vấn đề rất hạn chế, nhưng đủ để mang lại giá trị cho doanh nghiệp này. Tham vọng hơn nhiều, chúng tôi có thể muốn chính sách của mình được học trên 1 tập hợp hữu hạn các trường hợp để hoạt động tốt (tổng quát hóa) cho bất kỳ trường hợp lập trình tuyến tính số nguyên hỗn hợp "thực tế" nào. Điều này rất hữu ích nếu bạn đang kinh doanh bán các bộ giải lập trình tuyến tính số nguyên hỗn hợp & muốn chính sách phân nhánh hoạt động tốt cho càng nhiều khách hàng của mình càng tốt. Trong cả 2 trường hợp, khái quát hóa áp dụng cho các trường hợp mà người triển khai thuật toán không biết. Đây chỉ là những trường hợp mà chúng tôi quan tâm; trường hợp được sử dụng để đào tạo đã được giải quyết. Chủ đề về phân phối xác suất của các trường hợp cũng xuất hiện 1 cách tự nhiên trong tối ưu hóa lập trình ngẫu nhiên, trong đó sự không chắc chắn về vấn đề được mô hình hóa thông qua phân phối xác suất. Tạo kịch bản, 1 phương pháp thiết yếu để giải quyết loại chương trình tối ưu hóa này, đòi hỏi phải lấy mẫu từ phân phối này & giải quyết bài toán liên quan nhiều lần. Nair, Dvijotham, Dunning, & Vinyals (2018) đã tận dụng quy trình lặp lại này để học 1 mô hình đầu cuối nhằm giải quyết vấn đề. Mô hình của họ bao gồm 1 tìm kiếm cục bộ & 1 chính sách cải tiến cục bộ & được huấn luyện thông qua RL. Ở đây, khái quát hóa có nghĩa là: trong quá trình khái quát hóa kịch bản, tìm kiếm đã học sẽ vượt trội hơn các phương pháp khác, do đó mang lại 1 thuật toán quy hoạch ngẫu nhiên tổng thể nhanh hơn. Tóm lại, *học mà không có khái quát hóa thì vô nghĩa!*

When policy generalizes to other problem instances, it is no longer a problem if training requires additional computation for solving problem instances because, learning can be decoupled from solving as it can be done offline. This setting is promising as it could give a policy to use out of box for similar instances, while keeping learning problem to be handled beforehand while remaining hopefully reasonable. When model learned is a simple mapping, as is case in Sect. 3.2.2, generalization to new instances, as previously explained, can be easily understood. However, when learning sequential decisions, as in Sect. 3.2.3, there are intricate levels of generalization. Said: want policy to generalize to new instances, but policy also needs to generalize to internal states of algorithm for a single instance, even if model can be learned from complete optimization trajectories, as formulated by (8). Indeed, complex algorithms can have unexpected sources of randomness, even if they are designed to be deterministic. E.g., a numerical approximation may perform differently if version of some underlying numerical library is changed or because of asynchronous computing, e.g. when using Graphical Processing Units (Nagarajan, Warnell, & Stone, 2019). Furthermore, even if we can achieve perfect replicability, do not want branching policy to break if some other parameters of solver are set (slight) differently. At very least, want policy to be robust to choice of random seed present in many algorithms, including mixed-integer linear programming solvers. These parameters can therefore be modeled as random variables. Because of these nested levels of generalization, 1 appealing way to think about training data from multiple instances is like separate tasks of a multi-task learning setting. Different tasks have underlying aspects in common, & they may also have their own peculiar quirks. 1 way to learn a single policy that generalizes within a distribution of instances: take advantage of these commonalities. Generalization in RL remains a challenging topic, probably because of fuzzy distinction between a multi-task setting, & a large environment encompassing of all of tasks.

– Khi chính sách được khái quát hóa cho các trường hợp vấn đề khác, sẽ không còn là vấn đề nếu việc đào tạo yêu cầu tính toán bổ sung để giải quyết các trường hợp vấn đề, bởi vì việc học có thể được tách rời khỏi việc giải quyết vì nó có thể được thực hiện ngoại tuyến. Thiết lập này rất hứa hẹn vì nó có thể cung cấp 1 chính sách để sử dụng ngay cho các trường hợp tương tự, đồng thời vẫn giữ cho vấn đề học được xử lý trước mà vẫn đảm bảo tính hợp lý. Khi mô hình học được là 1 phép ánh xạ đơn giản, như trường hợp trong Mục 3.2.2, việc khái quát hóa cho các trường hợp mới, như đã giải thích trước đó, có thể dễ dàng hiểu được. Tuy nhiên, khi học các quyết định tuần tự, như trong Mục 3.2.3, có những cấp độ khái quát hóa phức tạp. Nói cách khác: muốn chính sách khái quát hóa cho các trường hợp mới, nhưng chính sách cũng cần khái quát hóa cho các trạng thái nội tại của thuật toán cho 1 trường hợp duy nhất, ngay cả khi mô hình có thể được học từ các quỹ đạo tối ưu hóa hoàn chỉnh, như được xây dựng bởi (8). Thật vậy, các thuật toán phức tạp có thể có các nguồn ngẫu nhiên không mong muốn, ngay cả khi chúng được thiết kế để mang tính xác định. Ví dụ: 1 phép xấp xỉ số có thể hoạt động khác nhau nếu phiên bản của 1 số thư viện số cơ bản bị thay đổi hoặc do tính toán không đồng bộ, ví dụ: khi sử dụng Đơn vị Xử lý Đồ họa (Nagarajan, Warnell, & Stone, 2019). Hơn nữa, ngay cả khi chúng ta có thể đạt được khả năng sao chép hoàn hảo, chúng ta cũng không muốn chính sách phân nhánh bị phá vỡ nếu 1 số tham số khác của bộ giải được đặt (hơi) khác biệt. Ít nhất, chúng ta muốn chính sách phải mạnh mẽ đối với việc lựa chọn hạt giống ngẫu nhiên có trong nhiều thuật toán, bao gồm cả bộ giải quy hoạch tuyến tính hỗn hợp số nguyên. Do đó, các tham số này có thể được mô hình hóa dưới dạng các biến ngẫu nhiên. Do các mức độ khái quát hóa lồng nhau này, 1 cách hấp dẫn để nghĩ về dữ liệu huấn luyện từ nhiều trường hợp giống như các tác vụ riêng biệt của 1 môi trường học tập đa tác vụ. Các tác vụ khác nhau có những khía

cạnh cơ bản chung, & chúng cũng có thể có những điểm kỳ quặc riêng. Một cách để học 1 chính sách duy nhất có thể khái quát hóa trong 1 phân phối các trường hợp: hãy tận dụng những điểm chung này. Khái quát hóa trong RL vẫn là 1 chủ đề đầy thách thức, có thể là do sự phân biệt mơ hồ giữa 1 môi trường đa tác vụ, & 1 môi trường rộng lớn bao gồm tất cả các tác vụ.

Choosing how ambitious one should be in defining characteristics of distribution is a hard question. E.g., if Montreal company expands its business to other cities, should they be considered as separate distributions, & learn 1 branching policy per city, or only a single one? Maybe 1 per continent? Generalization to a larger variety of instances is challenging & requires more advanced & expensive learning algorithms. Learning an array of ML models for different distributions associated with a same task means of course more models to train, maintain, & deploy. Same goes with traditional combinatorial optimization algorithms, an mixed-integer linear programming solver on its own is not best performing algorithm to solve TSP, but it works across all mixed-integer linear programming problems. Too early to provide insights about how broad considered distributions should be, given limited literature in field. For scholars generating synthetic distributions, 2 intuitive axes of investigation are "structure" & "size". A TSP & a scheduling problem seem to have fairly different structure, & one can think of 2 planar euclidean TSP to be way more similar. Still, 2 of these TSP can have dramatically different sizes (number of nodes). E.g., Gasse et al. (2019) assess their methodology independently on 3 distributions. Each training dataset has a specific problem structure (set covering, combinatorial auction, & capacitated facility location), & a fixed problem size. Problem instance generators used are state-of-art & representative of real-world instances. Nonetheless, when they evaluate their learned algorithm, authors push test distributions to larger sizes. Idea behind this: gauge if model learned is able to generalize to a larger, more practical, distribution, or only perform well on restricted distribution of problems of same size. Answer is largely affirmative.

– Việc lựa chọn mức độ tham vọng khi xác định các đặc điểm của phân phối là 1 câu hỏi khó. Ví dụ, nếu 1 công ty ở Montreal mở rộng hoạt động kinh doanh sang các thành phố khác, liệu chúng có nên được coi là các phân phối riêng biệt, & học 1 chính sách phân nhánh cho mỗi thành phố hay chỉ 1 chính sách duy nhất? Có thể là 1 cho mỗi lục địa? Việc khái quát hóa cho nhiều trường hợp hơn là 1 thách thức & đòi hỏi các thuật toán học tiên tiến hơn & tốn kém hơn. Học 1 mảng các mô hình ML cho các phân phối khác nhau liên quan đến cùng 1 nhiệm vụ tất nhiên có nghĩa là cần nhiều mô hình hơn để đào tạo, duy trì, & triển khai. Tương tự như các thuật toán tối ưu hóa tổ hợp truyền thống, 1 bộ giải quy hoạch tuyến tính số nguyên hỗn hợp tự nó không phải là thuật toán có hiệu suất tốt nhất để giải TSP, nhưng nó hoạt động trên tất cả các vấn đề quy hoạch tuyến tính số nguyên hỗn hợp. Còn quá sớm để cung cấp thông tin chi tiết về mức độ rộng rãi của các phân phối được xem xét, do tài liệu trong lĩnh vực này còn hạn chế. Đối với các học giả tạo ra các phân phối tổng hợp, 2 trục điều tra trực quan là "cấu trúc" & "kích thước". Một TSP & 1 bài toán lập lịch dường như có cấu trúc khá khác nhau, & người ta có thể nghĩ đến 2 TSP euclidean phẳng giống nhau hơn nhiều. Tuy nhiên, 2 trong số các TSP này có thể có kích thước khác nhau đáng kể (số lượng nút). Ví dụ: Gasse & cộng sự (2019) đánh giá phương pháp luận của họ 1 cách độc lập trên 3 phân phối. Mỗi tập dữ liệu đào tạo có 1 cấu trúc bài toán cụ thể (phủ tập hợp, đấu giá tổ hợp, & vị trí cơ sở có năng lực), & kích thước bài toán cố định. Các trình tạo thể hiện bài toán được sử dụng là công nghệ tiên tiến & đại diện cho các thể hiện trong thế giới thực. Tuy nhiên, khi họ đánh giá thuật toán đã học của mình, các tác giả đẩy các phân phối thử nghiệm lên kích thước lớn hơn. Ý tưởng đằng sau điều này: đánh giá xem mô hình đã học có thể khái quát hóa thành phân phối lớn hơn, thực tế hơn hay chỉ hoạt động tốt trên phân phối hạn chế của các bài toán có cùng kích thước. Câu trả lời phần lớn là khẳng định.

- 4.4. Single instance learning. An edge case that we have not much discussed yet is single instance learning framework. This might be case e.g. for planning design of a single factory. Factory would only be built once, with very peculiar requirements, & planners are not interested to relate this to other problems. In this case, one can make as many runs (episodes) & as many calls to a potential expert or simulator as one wants, but ultimately one only cares about solving this 1 instance. Learning a policy for a single instance should require a simpler ML model, which could thus require less training examples. Nonetheless, in single instance case, one learns policy from scratch at every new instance, actually incorporating learning (not learned models but really learning process itself) into end algorithm. I.e., starting timer at beginning of learning & competing with other solvers to get solution fastest (or get best results within a time limit). This is an edge scenarios that can only be employed in setting of Sect. 3.2.3, where ML is embedded *inside* a combinatorial optimization algorithm; otherwise there would be only 1 training example! There is therefore no notion of generalization to other problem instances, so (6) is not learning problem being solved. Nonetheless, model still needs to generalize to *unseen states* of algorithm. Indeed, if model was learned from all states of algorithm that are needed to solve problem, then problem is already solved at training time & learning is therefore fruitless. This is methodology followed by Khalil et al. (2016), introduced in Sect. 3.1.1, to learn an instance-specific branching policy. Policy is learned from strong-branching at top of branch-&-bound tree, but needs to generalize to state of algorithm at bottom of tree, where it is used. However, as for all combinatorial optimization algorithms, a fair comparison to another algorithm can only be done on an independent dataset of instances, as in (4). This is because through human trials & errors, data used when building algorithm leaks into design of algorithm, even without explicit learning components.

  – Học tập cá thể đơn lẻ. Một trường hợp ngoại lệ mà chúng ta chưa thảo luận nhiều là khuôn khổ học tập cá thể đơn lẻ. Ví dụ, đây có thể là trường hợp lập kế hoạch thiết kế 1 nhà máy đơn lẻ. Nhà máy chỉ được xây dựng 1 lần, với các yêu cầu rất đặc thù, & các nhà lập kế hoạch không quan tâm đến việc liên hệ điều này với các vấn đề khác. Trong trường hợp này, người ta có thể thực hiện bao nhiêu lần chạy (tập) & bao nhiêu cuộc gọi đến 1 chuyên gia hoặc trình mô phỏng tiềm năng tùy thích, nhưng cuối cùng người ta chỉ quan tâm đến việc giải quyết 1 cá thể này. Việc học 1 chính sách cho 1 cá thể đơn lẻ sẽ yêu cầu 1 mô hình ML đơn giản hơn, do đó có thể cần ít ví dụ huấn luyện hơn. Tuy nhiên, trong trường hợp cá thể đơn lẻ, người ta học chính sách từ đầu tại mỗi cá thể mới, thực sự kết hợp việc học (không phải các mô hình đã học mà

thực sự là chính quá trình học) vào thuật toán cuối cùng. Tức là, bắt đầu bộ đếm thời gian khi bắt đầu học & cạnh tranh với các trình giải khác để có được giải pháp nhanh nhất (hoặc đạt được kết quả tốt nhất trong giới hạn thời gian). Đây là 1 kịch bản ngoại lệ chỉ có thể được sử dụng trong bối cảnh của Mục 3.2.3, trong đó ML được nhúng *bên trong* 1 thuật toán tối ưu hóa tổ hợp; nếu không thì sẽ chỉ có 1 ví dụ huấn luyện! Do đó, không có khái niệm khái quát hóa cho các trường hợp vấn đề khác, do đó (6) không phải là vấn đề học tập đang được giải quyết. Tuy nhiên, mô hình vẫn cần khái quát hóa thành *các trạng thái chưa thấy* của thuật toán. Thật vậy, nếu mô hình được học từ tất cả các trạng thái của thuật toán cần thiết để giải quyết vấn đề, thì vấn đề đã được giải quyết tại thời điểm huấn luyện & do đó việc học tập là vô ích. Đây là phương pháp luận do Khalil & cộng sự (2016) áp dụng, được giới thiệu trong Phần 3.1.1, để học chính sách phân nhánh cụ thể cho từng trường hợp. Chính sách được học từ phân nhánh mạnh ở đầu cây có nhánh &-bound, nhưng cần khái quát hóa thành trạng thái của thuật toán ở cuối cây, nơi nó được sử dụng. Tuy nhiên, đối với tất cả các thuật toán tối ưu hóa tổ hợp, việc so sánh công bằng với 1 thuật toán khác chỉ có thể được thực hiện trên 1 tập dữ liệu độc lập của các trường hợp, như trong (4). Điều này là do thông qua các lần thử & lỗi của con người, dữ liệu được sử dụng khi xây dựng thuật toán bị rò rỉ vào thiết kế thuật toán, ngay cả khi không có các thành phần học tập rõ ràng.

- 4.5. Fine tuning & meta-learning. A compromise between instance-specific learning & learning a generic policy is what we typically have in multitask learning: some parameters are shared across tasks & some are specific to each task. A common way to do that (in transfer learning scenario): start from a generic policy & then adapt it to particular instance by a form of *fine-tuning* procedure: training proceeds in 2 stages, 1st training generic policy across many instances from same distribution, & then continuing training on examples associated with a given instance on which we are hoping to get more specialized & accurate predictions.

  – Tinh chỉnh & siêu học. Một sự dung hòa giữa học tập theo từng trường hợp & học tập theo chính sách chung là điều chúng ta thường có trong học tập đa tác vụ: 1 số tham số được chia sẻ giữa các tác vụ & 1 số tham số dành riêng cho từng tác vụ. Một cách phổ biến để thực hiện điều đó (trong kịch bản học chuyển giao): bắt đầu từ 1 chính sách chung & sau đó điều chỉnh nó cho từng trường hợp cụ thể bằng 1 hình thức *tinh chỉnh*: quá trình huấn luyện diễn ra theo 2 giai đoạn, đầu tiên là huấn luyện chính sách chung trên nhiều trường hợp từ cùng 1 phân phối, & sau đó tiếp tục huấn luyện trên các ví dụ liên quan đến 1 trường hợp nhất định mà chúng ta hy vọng sẽ có được những dự đoán chuyên biệt & chính xác hơn.

  ML advances in areas of meta-learning & transfer learning are particularly interesting to consider here. Meta-learning considers 2 levels of optimization: inner loop trains parameters of a model on training set in a way that depends on meta-parameters, which are themselves optimized in an outer loop (i.e., obtaining a gradient for each completed inner-loop training or update). When outer loop's objective function is performance on a validation set, end up training a system so that it will generalize well. This can be a successful strategy for generalizing from very few examples if have access to many such training tasks. It is related to transfer learning, where want that what hs been learned in 1 or many tasks helps improve generalization on another. These approaches can help rapidly adapt to a new problem, which would be useful in context of solving many mixed-integer linear programming instances, seen as many related tasks.

  – Tiến bộ ML trong các lĩnh vực học siêu dữ liệu & học chuyển giao đặc biệt thú vị để xem xét ở đây. Học siêu dữ liệu xem xét 2 cấp độ tối ưu hóa: vòng lặp bên trong huấn luyện các tham số của 1 mô hình trên tập huấn luyện theo cách phụ thuộc vào các siêu dữ liệu, bản thân chúng được tối ưu hóa trong vòng lặp bên ngoài (tức là, thu được 1 gradient cho mỗi lần huấn luyện hoặc cập nhật vòng lặp bên trong đã hoàn thành). Khi hàm mục tiêu của vòng lặp bên ngoài là hiệu suất trên 1 tập xác thực, hãy kết thúc việc huấn luyện 1 hệ thống để nó có thể khái quát hóa tốt. Đây có thể là 1 chiến lược thành công để khái quát hóa từ rất ít ví dụ nếu có quyền truy cập vào nhiều tác vụ huấn luyện như vậy. Nó liên quan đến học chuyển giao, trong đó muốn rằng những gì đã học được trong 1 hoặc nhiều tác vụ giúp cải thiện khả năng khái quát hóa trong tác vụ khác. Các cách tiếp cận này có thể giúp nhanh chóng thích ứng với 1 vấn đề mới, điều này sẽ hữu ích trong bối cảnh giải quyết nhiều trường hợp lập trình tuyến tính số nguyên hỗn hợp, được coi là nhiều tác vụ liên quan.

  To stay with branching example on mixed-integer linear programming, one may not want policy to perform well out of box on new instances (from given distribution). Instead, one may want to learn a policy offline that can be adapted to a new instance in a few training steps, every time it is given one. Similar topics have been explored in context of automatic configuration tools. Fitzgerald, Malitsky, O'Sullivan, & Tierney (2014) study automatic configuration in lifelong learning context (a form of sequential transfer learning). Automatic configuration algorithm is augmented with a set of previous configurations that are prioritized on any new problem instance. A score reflecting past performances is kept along every configuration. It is designed to retain configurations that performed well in past, while letting new ones a chance to be properly evaluated. Automatic configuration optimization algorithm used by Lindauer & Hutter (2018) requires training an empirical cost model mapping Cartesian product of parameter configurations & problem instances to expected algorithmic performance. Such a model is usually learned for every cluster of problem instance that requires configuring. Instead, when presented with a new cluster, authors combine previously learned cost models & new one to build an ensemble model. As done by Fitzgerald et al. (2014), authors also build a set of previous configurations to prioritize, using an empirical cost model to fill missing data. This setting, which is more general than not performing any adaptation of policy, has potential for better generalization. Once again, scale on which this is applied can vary depending on ambition. One can transfer on very similar instances, or learn a policy that transfers to a vast range of instances.

  – Để tiếp tục với ví dụ về phân nhánh trong lập trình tuyến tính số nguyên hỗn hợp, người ta có thể không muốn chính sách hoạt động tốt ngay lập tức trên các trường hợp mới (từ phân phối cho trước). Thay vào đó, người ta có thể muốn học 1 chính sách ngoại tuyến có thể được điều chỉnh cho 1 trường hợp mới trong 1 vài bước huấn luyện, mỗi khi được đưa vào. Các chủ đề tương tự đã được khám phá trong bối cảnh của các công cụ cấu hình tự động. Fitzgerald, Malitsky, O'Sullivan, & Tierney (2014) nghiên cứu cấu hình tự động trong bối cảnh học tập suốt đời (một dạng học chuyển giao tuần tự). Thuật toán cấu hình tự động được bổ sung bằng 1 tập hợp các cấu hình trước đó được ưu tiên trên bất kỳ trường hợp bài toán

mới nào. Một điểm số phản ánh hiệu suất trong quá khứ được lưu giữ trong mỗi cấu hình. Nó được thiết kế để giữ lại các cấu hình hoạt động tốt trong quá khứ, đồng thời cho phép các cấu hình mới có cơ hội được đánh giá đúng cách. Thuật toán tối ưu hóa cấu hình tự động được Lindauer & Hutter (2018) sử dụng yêu cầu huấn luyện 1 mô hình chi phí thực nghiệm, ánh xạ tích Descartes của các cấu hình tham số & trường hợp bài toán với hiệu suất thuật toán dự kiến. Một mô hình như vậy thường được học cho mọi cụm trường hợp bài toán cần cấu hình. Thay vào đó, khi được trình bày với 1 cụm mới, các tác giả kết hợp các mô hình chi phí đã học trước đó & 1 mô hình mới để xây dựng 1 mô hình tổng thể. Như Fitzgerald & cộng sự (2014) đã thực hiện, các tác giả cũng xây dựng 1 tập hợp các cấu hình trước đó để ưu tiên, sử dụng mô hình chi phí thực nghiệm để lấp đầy dữ liệu còn thiếu. Thiết lập này, mang tính tổng quát hơn so với việc không thực hiện bất kỳ điều chỉnh chính sách nào, có tiềm năng khái quát hóa tốt hơn. Một lần nữa, quy mô áp dụng điều này có thể thay đổi tùy thuộc vào tham vọng. Người ta có thể chuyển đổi trên các trường hợp rất giống nhau, hoặc học 1 chính sách có thể chuyển đổi sang 1 phạm vi rộng lớn các trường hợp.

Meta-learning algorithms were 1st introduced in 1990s (Bengio, Bengio, Cloutier, & Gecsei, 1991; Schmidhuber, 1992; Thrun & Pratt, 1998) & have since then become particularly popular in ML, including, but not limited to, learning a gradient update rule (Andrychowicz et al., 2016; Hochreiter, Younger, & Conwell, 2001), few shot learning (Ravi & Larochelle, 2017), & multitask RL (Finn, Abbeel, & Levine, 2017).

– Thuật toán siêu học được giới thiệu lần đầu tiên vào những năm 1990 (Bengio, Bengio, Cloutier, & Gecsei, 1991; Schmidhuber, 1992; Thrun & Pratt, 1998) & kể từ đó đã trở nên đặc biệt phổ biến trong ML, bao gồm nhưng không giới hạn ở việc học quy tắc cập nhật gradient (Andrychowicz & cộng sự, 2016; Hochreiter, Younger, & Conwell, 2001), học vài lần (Ravi & Larochelle, 2017), & RL đa nhiệm (Finn, Abbeel, & Levine, 2017).

- 4.6. Other metrics. Other metrics from process of learning itself are also relevant, e.g. how fast learning process is, sample complexity (number of examples required to properly fit model), etc. As opposed to metrics suggested earlier in this sect, these metrics provide us with information not about final performance, but about offline computation or number of training examples required to obtain desired policy. This information is, of course, useful to calibrate effort in integrating ML into combinatorial optimization algorithms.

  – Các số liệu khác từ chính quá trình học cũng có liên quan, ví dụ như tốc độ học, độ phức tạp của mẫu (số lượng ví dụ cần thiết để mô hình phù hợp), v.v. Trái ngược với các số liệu đã đề xuất trước đó trong phần này, các số liệu này không cung cấp cho chúng ta thông tin về hiệu suất cuối cùng, mà là về tính toán ngoại tuyến hoặc số lượng ví dụ đào tạo cần thiết để đạt được chính sách mong muốn. Tất nhiên, thông tin này hữu ích để hiệu chỉnh nỗ lực tích hợp ML vào các thuật toán tối ưu hóa tổ hợp.

- 5. Methodology. In prev sect, have detailed theoretical learning framework of using ML in combinatorial optimization algorithms. Hence, provide some additional discussion broadening some previously made claims.

  – Trong phần trước, chúng tôi đã trình bày chi tiết về khuôn khổ học tập lý thuyết về việc sử dụng ML trong các thuật toán tối ưu hóa tổ hợp. Do đó, chúng tôi sẽ cung cấp thêm 1 số thảo luận để mở rộng 1 số tuyên bố đã được đưa ra trước đó.

  - 5.1. Demonstration & experience. In order to learn a policy, have highlighted 2 methodologies: demonstration, where expected behavior is shown by an expert or oracle (sometimes at a considerable computational cost), & experience, where policy is learned through trial & error with a reward signal.

    – Trình diễn & kinh nghiệm. Để học 1 chính sách, đã nêu bật 2 phương pháp: trình diễn, trong đó hành vi mong đợi được thể hiện bởi 1 chuyên gia hoặc nhà tiên tri (đôi khi tốn kém chi phí tính toán đáng kể), & kinh nghiệm, trong đó chính sách được học thông qua thử nghiệm & sai sót với tín hiệu thưởng.

    In demonstration setting, performance of learned policy is bounded by expert, which is a limitation when expert is not optimal. More precisely, without a reward signal, imitation policy can only hope to marginally outperform expert (e.g. because learner can reduce variance of answers across similarly-performing experts). Better learning, closer performance of learner to expert's. I.e., imitation alone should be used only if it is significantly faster than expert to compute policy. Furthermore, performance of learned policy may not generalize well to unseen examples & small variations of task & may be unstable due to accumulation of errors. This is because in (9), data was collected according to expert policy π_e, but when run over multiple repeated decisions, distribution of states becomes that of learned policy. Some downsides of supervised (imitation) learning can be overcome with more advanced algorithms, including active methods to query expert as an oracle to improve behavior in uncertain states. Part of imitation learning presented here is limited compared to current literature in ML.

    – Trong bối cảnh trình diễn, hiệu suất của chính sách đã học bị giới hạn bởi chuyên gia, đây là 1 hạn chế khi chuyên gia không tối ưu. Chính xác hơn, nếu không có tín hiệu thưởng, chính sách bắt chước chỉ có thể hy vọng vượt trội hơn chuyên gia 1 chút (ví dụ: vì người học có thể giảm phương sai của câu trả lời giữa các chuyên gia có hiệu suất tương tự). Học tốt hơn, hiệu suất của người học gần hơn với chuyên gia. Tức là, chỉ nên sử dụng bắt chước nếu nó nhanh hơn đáng kể so với chuyên gia trong việc tính toán chính sách. Hơn nữa, hiệu suất của chính sách đã học có thể không tổng quát hóa tốt với các ví dụ chưa biết & các biến thể nhỏ của nhiệm vụ & có thể không ổn định do tích tụ lỗi. Điều này là do trong (9), dữ liệu được thu thập theo chính sách chuyên gia π_e, nhưng khi chạy qua nhiều quyết định lặp lại, phân phối trạng thái trở thành phân phối của chính sách đã học. Một số nhược điểm của học có giám sát (bắt chước) có thể được khắc phục bằng các thuật toán tiên tiến hơn, bao gồm các phương pháp chủ động để truy vấn chuyên gia như 1 oracle để cải thiện hành vi trong các trạng thái không chắc chắn. Một phần của học bắt chước được trình bày ở đây còn hạn chế so với các tài liệu hiện có về ML.

On contrary, with a reward, algorithm learns to optimize for that signal & can potentially outperform any expert, at cost of a much longer training time. Learning from a reward signal (experience) is also more flexible when multiple decisions are (almost) equally good in comparison with an expert that would favor 1 (arbitrary) decision. Experience is not without flaws. In case where policies are approximated (e.g., with a neural network), learning process may get stuck around poor solutions if exploration is not sufficient or solutions which do not generalize well are found. Furthermore, it may not always be straightforward to define a reward signal. E.g., sparse rewards may be augmented using reward shaping or a curriculum in order to value intermediate accomplishments (see Sect. 2.2).

– Ngược lại, với phần thưởng, thuật toán học cách tối ưu hóa cho tín hiệu đó & có khả năng vượt trội hơn bất kỳ chuyên gia nào, với chi phí thời gian đào tạo dài hơn nhiều. Học từ tín hiệu phần thưởng (kinh nghiệm) cũng linh hoạt hơn khi nhiều quyết định (gần như) tốt như nhau so với 1 chuyên gia chỉ thiên về 1 quyết định (tùy ý). Kinh nghiệm không phải là không có sai sót. Trong trường hợp các chính sách được xấp xỉ (ví dụ: với mạng nơ-ron), quá trình học có thể bị mắc kẹt xung quanh các giải pháp kém nếu việc khám phá không đủ hoặc các giải pháp không có khả năng khái quát hóa tốt được tìm thấy. Hơn nữa, việc xác định tín hiệu phần thưởng không phải lúc nào cũng đơn giản. Ví dụ: phần thưởng thưa thớt có thể được tăng cường bằng cách sử dụng định hình phần thưởng hoặc chương trình giảng dạy để đánh giá cao các thành tích trung gian (xem Phần 2.2).

Often, it is a good idea to start learning from demonstrations by an expert, then refine policy using experience & a reward signal. This is what was done in original AlphaGo paper (Silver et al., 2016), where human knowledge is combined with RL. Reader is referred to Hussein, Gaber, Elyan, & Jayne (2017) for a survey on imitation learning covering most of discussion in this sect.

– Thông thường, nên bắt đầu học hỏi từ các bài trình diễn của chuyên gia, sau đó tinh chỉnh chính sách bằng kinh nghiệm & tín hiệu thưởng. Đây là những gì đã được thực hiện trong bài báo AlphaGo gốc (Silver & cộng sự, 2016), trong đó kiến thức của con người được kết hợp với RL. Độc giả có thể tham khảo Hussein, Gaber, Elyan, & Jayne (2017) để biết thêm về khảo sát về học tập bắt chước, bao gồm hầu hết các thảo luận trong lĩnh vực này.

- 5.2. Partial observability. Mentioned in Sect. 2.2 that sometimes states of an Markov decision process are not fully observed & Markov property does not hold, i.e., probability of next observation, conditioned on current observation & action, is not equal to probability of next observation, conditioned on all past observations & actions. An immediate example of this can be found in any environment simulating physics: a single frame/image of such an environment is not sufficient to grasp notions e.g. velocity & is therefore not sufficient to properly estimate future trajectory of objects. Turn out: on real applications, partial observability is closer to norm than to exception, either because one does not have access to a true state of environment, or because it is not computationally tractable to represent & needs to be approximated. A straightforward way to tackle problem: compress all previous observations using a RNN. This can be applied in imitation learning setting, as well as in RL, e.g., by learning a recurrent policy (Wierstra, Förster, Peters, & Schmidhuber, 2010).

  – Khả năng quan sát 1 phần. Như đã đề cập trong Phần 2.2, đôi khi các trạng thái của quy trình quyết định Markov không được quan sát đầy đủ & Tính chất Markov không đúng, tức là xác suất của quan sát tiếp theo, phụ thuộc vào quan sát hiện tại & hành động, không bằng xác suất của quan sát tiếp theo, phụ thuộc vào tất cả các quan sát trước đó & hành động. Một ví dụ trực tiếp về điều này có thể được tìm thấy trong bất kỳ môi trường nào mô phỏng vật lý: 1 ảnh khung hình duy nhất của 1 môi trường như vậy là không đủ để nắm bắt các khái niệm, ví dụ như vận tốc & do đó không đủ để ước tính chính xác quỹ đạo tương lai của các vật thể. Hóa ra: trong các ứng dụng thực tế, khả năng quan sát 1 phần gần với chuẩn mực hơn là ngoại lệ, hoặc vì người ta không có quyền truy cập vào trạng thái thực của môi trường, hoặc vì không thể tính toán để biểu diễn & cần phải được xấp xỉ. Một cách đơn giản để giải quyết vấn đề: nén tất cả các quan sát trước đó bằng RNN. Điều này có thể được áp dụng trong bối cảnh học tập mô phỏng cũng như trong RL, ví dụ, bằng cách học 1 chính sách lặp lại (Wierstra, Förster, Peters, & Schmidhuber, 2010).

  How does this apply in case where we want to learn a policy function making decisions for a combinatorial optimization algorithm? On 1 hand, one has full access to state of algorithm because it is represented in exact mathematical concepts, e.g. constraints, cuts, solutions, branch-&-bound tree, etc. On other hand, these states can be exponentially large. This is an issue in terms of computations & generalization. Indeed, if one does want to solve problems quickly, one needs to have a policy that is also fast to compute, especially if it is called frequently as is case for, say, branching decisions. Furthermore, considering too high-dimensional states is also a statistical problem for learning, as it may dramatically increase required number of samples, decrease learning speed, or fail altogether. Hence, necessary to keep these aspects in mind while experimenting with different representations of data.

  – Điều này áp dụng như thế nào trong trường hợp chúng ta muốn học 1 hàm chính sách đưa ra quyết định cho 1 thuật toán tối ưu hóa tổ hợp? Một mặt, người ta có toàn quyền truy cập vào trạng thái của thuật toán vì nó được biểu diễn bằng các khái niệm toán học chính xác, ví dụ: ràng buộc, phép cắt, nghiệm, cây nhánh &-bound, v.v. Mặt khác, các trạng thái này có thể lớn theo cấp số nhân. Đây là 1 vấn đề về mặt tính toán & khái quát hóa. Thật vậy, nếu muốn giải quyết vấn đề nhanh chóng, người ta cần có 1 chính sách cũng có thể tính toán nhanh, đặc biệt nếu nó được gọi thường xuyên như trường hợp, chẳng hạn như các quyết định phân nhánh. Hơn nữa, việc xem xét các trạng thái có quá nhiều chiều cũng là 1 vấn đề thống kê đối với việc học, vì nó có thể làm tăng đáng kể số lượng mẫu cần thiết, làm giảm tốc độ học hoặc hoàn toàn thất bại. Do đó, cần phải lưu ý những khía cạnh này khi thử nghiệm với các biểu diễn dữ liệu khác nhau.

- 5.3. Exactness & approximation. In different examples we have surveyed, ML is used in both exact & heuristic frameworks, e.g. Baltean-Lugojan et al. (2018) & Larsen et al. (2018), resp. Getting output of an ML model to respect advanced types of constraints is a hard task. In order to build exact algorithms with ML components, necessary to apply ML where all possible decisions are valid. Using only ML as surveyed in Sect. 3.2.1 cannot give any optimality guarantee, & only weak feasibility

guarantees (see Sect. 6.1). However, applying ML to select or parametrize a combinatorial optimization algorithm as in Sect. 3.3.2 will keep exactness if all possible choices that ML discriminate lead to complete algorithms. Finally, in case of repeated interactions between ML & combinatorial optimization surveyed in Sect. 3.2.3, all possible decisions must be valid. E.g., in case of mixed-integer linear programming, this includes branching *among fractional variables* of linear programming relaxation, selecting node to explore *among open branching nodes* (He et al., 2014), deciding on frequency to run heuristics on branch-&-bound nodes (Khalil et al., 2017b), selecting cutting planes *among valid inequalities* (Baltean-Lugojan et al., 2018), removing previous cutting planes *if they are not original constraints or branching decision*, etc. A counterexample can be found in work of Hottung et al. (2017), presented in Sect. 3.1.1. In their branch-&-bound framework, bounding is performed by an approximate ML model that can overestimate lower bounds, resulting in invalid pruning. Resulting algorithm is therefore not an exact one.

– Độ chính xác & xấp xỉ. Trong các ví dụ khác nhau mà chúng tôi đã khảo sát, ML được sử dụng trong cả khuôn khổ chính xác & heuristic, ví dụ: Baltean-Lugojan & cộng sự (2018) & Larsen & cộng sự (2018), tương ứng. Việc đưa đầu ra của mô hình ML tuân thủ các loại ràng buộc nâng cao là 1 nhiệm vụ khó khăn. Để xây dựng các thuật toán chính xác với các thành phần ML, cần áp dụng ML khi tất cả các quyết định khả thi đều hợp lệ. Chỉ sử dụng ML như đã khảo sát trong Phần 3.2.1 không thể đưa ra bất kỳ đảm bảo tối ưu nào, & chỉ có các đảm bảo khả thi yếu (xem Phần 6.1). Tuy nhiên, việc áp dụng ML để lựa chọn hoặc tham số hóa thuật toán tối ưu hóa tổ hợp như trong Phần 3.3.2 sẽ giữ được độ chính xác nếu tất cả các lựa chọn khả thi mà ML phân biệt đều dẫn đến các thuật toán hoàn chỉnh. Cuối cùng, trong trường hợp có sự tương tác lặp lại giữa ML & tối ưu hóa tổ hợp như đã khảo sát trong Phần 3.2.3, tất cả các quyết định khả thi phải hợp lệ. Ví dụ, trong trường hợp lập trình tuyến tính số nguyên hỗn hợp, điều này bao gồm việc phân nhánh *giữa các biến phân số* của quá trình nới lỏng lập trình tuyến tính, chọn nút để khám phá *giữa các nút phân nhánh mở* (He et al., 2014), quyết định tần suất chạy thuật toán tìm kiếm trên các nút ràng buộc nhánh (Khalil et al., 2017b), chọn mặt phẳng cắt *giữa các bất đẳng thức hợp lệ* (Baltean-Lugojan et al., 2018), loại bỏ các mặt phẳng cắt trước đó *nếu chúng không phải là ràng buộc ban đầu hoặc quyết định phân nhánh*, v.v. Có thể tìm thấy 1 phản ví dụ trong công trình của Hottung et al. (2017), được trình bày trong Phần 3.1.1. Trong khuôn khổ ràng buộc nhánh của họ, việc ràng buộc được thực hiện bởi 1 mô hình ML gần đúng có thể ước tính quá cao các giới hạn dưới, dẫn đến việc cắt tỉa không hợp lệ. Do đó, thuật toán kết quả không phải là 1 thuật toán chính xác.

- 6. Challenges. In this sect, review some of algorithmic concepts previously introduced by taking viewpoint of their associated challenges.

  – Trong phần này, hãy xem xét lại 1 số khái niệm thuật toán đã được giới thiệu trước đó bằng cách xem xét những thách thức liên quan của chúng.

  - 6.1. Feasibility. In Sect. 3.2.1, pointed out how ML can be used to directly output solutions to optimization problems. Rather than learning solution, it would be more precise to say: algorithm is learning a *heuristic*,. As already repeatedly noted, learned algorithm does not give any guarantee in terms of optimality, but it is even more critical that feasibility is not guaranteed either. Indeed, do not know how far output of heuristic is from optimal solution, or if it even respects constraints of problem. This can be case for every heuristic & issue can be mitigated by using heuristic within an exact optimization algorithm (e.g., branch & bound).

    – Khả thi. Trong Phần 3.2.1, đã chỉ ra cách ML có thể được sử dụng để đưa ra trực tiếp các giải pháp cho các bài toán tối ưu hóa. Thay vì học giải pháp, sẽ chính xác hơn khi nói: thuật toán đang học 1 *heuristic*. Như đã lưu ý nhiều lần, thuật toán đã học không đảm bảo bất kỳ điều gì về tính tối ưu, nhưng điều quan trọng hơn là tính khả thi cũng không được đảm bảo. Thật vậy, không biết đầu ra của heuristic cách giải pháp tối ưu bao xa, hoặc liệu nó có tuân thủ các ràng buộc của bài toán hay không. Điều này có thể đúng với mọi heuristic & vấn đề có thể được giảm thiểu bằng cách sử dụng heuristic trong 1 thuật toán tối ưu hóa chính xác (ví dụ: nhánh & cận).

    Finding feasible solutions is not an easy problem (theoretically NP-hard for mixed-integer linear programming), but it is even more challenging in ML, especially by using neural networks. Indeed, trained with gradient descent, neural architectures must be designed carefully in order not to break differentiability. E.g., both pointer networks (Vinyals et al., 2015) & Sinkhorn layer (Emami & Ranka, 2018) are complex architectures used to make a network output a permutation, a constraint easy to satisfy when writing a classical combinatorial optimization heuristic.

    – Việc tìm kiếm các giải pháp khả thi không phải là 1 bài toán dễ dàng (về mặt lý thuyết là NP-khó đối với quy hoạch tuyến tính hỗn hợp), nhưng nó thậm chí còn khó khăn hơn trong ML, đặc biệt là khi sử dụng mạng nơ-ron. Thật vậy, khi được huấn luyện với gradient descent, kiến trúc nơ-ron phải được thiết kế cẩn thận để không phá vỡ tính khả vi. Ví dụ, cả mạng con trỏ (Vinyals & cộng sự, 2015) & lớp Sinkhorn (Emami & Ranka, 2018) đều là những kiến trúc phức tạp được sử dụng để tạo ra 1 mạng đầu ra là 1 hoán vị, 1 ràng buộc dễ thỏa mãn khi viết thuật toán tối ưu hóa tổ hợp cổ điển.

  - 6.2. Modeling. In ML, in general, & in DL, in particular, know some good prior for some given problems. E.g., know: a CNN is an architecture that will learn & generalize more easily than others on image data. Problems studied in combinatorial optimization are different from ones currently being addressed in ML, where most successful applications target natural signals. Architectures used to learn good policies in combinatorial optimization might be very different from what is currently used with DL. This might also be true in more subtle or unexpected ways: conceivable: in turn, optimization components of DL algorithms (say, modifications to stochastic gradient descent) could be different when DL is applied to combinatorial optimization context.

    – Mô hình hóa. Trong ML nói chung, & trong DL nói riêng, cần biết 1 số tiên nghiệm tốt cho 1 số bài toán nhất định. Ví dụ: biết: CNN là 1 kiến trúc sẽ học & tổng quát hóa dễ dàng hơn các kiến trúc khác trên dữ liệu hình ảnh. Các bài toán

được nghiên cứu trong tối ưu hóa tổ hợp khác với các bài toán hiện đang được giải quyết trong ML, nơi hầu hết các ứng dụng thành công đều nhắm đến các tín hiệu tự nhiên. Các kiến trúc được sử dụng để học các chính sách tốt trong tối ưu hóa tổ hợp có thể rất khác so với những kiến trúc hiện đang được sử dụng với DL. Điều này cũng có thể đúng theo những cách tinh tế hoặc bất ngờ hơn: có thể hình dung: ngược lại, các thành phần tối ưu hóa của thuật toán DL (ví dụ, các sửa đổi đối với giảm dần độ dốc ngẫu nhiên) có thể khác nhau khi DL được áp dụng vào bối cảnh tối ưu hóa tổ hợp.

Current DL already provides many techniques & architectures for tackling problems of interest in combinatorial optimization. As pointed out in Sect. 2.2, techniques e.g. parameter sharing made it possible for neural networks to process sequences of variable length with RNN or, more recently, to process graph structured data through GNN. Processing graph data is of uttermost importance in combinatorial optimization because many problems are formulated (represented) on graphs. For a very general example, Selsam et al. (2018) represent a satisfiability problem using a bipartite graph on variables & clauses. This can generalize to mixed-integer linear programming, where constraint matrix can be represented as adjacency matrix of a bipartite graph on variables & constraints, as done in Gasse et al. (2019).

– DL hiện tại đã cung cấp nhiều kỹ thuật & kiến trúc để giải quyết các vấn đề quan tâm trong tối ưu hóa tổ hợp. Như đã chỉ ra trong Phần 2.2, các kỹ thuật, ví dụ như chia sẻ tham số, đã giúp mạng nơ-ron có thể xử lý các chuỗi có độ dài biến đổi bằng RNN hoặc gần đây hơn là xử lý dữ liệu có cấu trúc đồ thị thông qua GNN. Việc xử lý dữ liệu đồ thị có tầm quan trọng tối đa trong tối ưu hóa tổ hợp vì nhiều bài toán được xây dựng (biểu diễn) trên đồ thị. Một ví dụ rất tổng quát, Selsam & cộng sự (2018) trình bày 1 bài toán khả năng thỏa mãn bằng cách sử dụng đồ thị 2 phần trên các biến & mệnh đề. Điều này có thể được khái quát hóa thành quy hoạch tuyến tính số nguyên hỗn hợp, trong đó ma trận ràng buộc có thể được biểu diễn dưới dạng ma trận kề của đồ thị 2 phần trên các biến & ràng buộc, như đã được thực hiện trong Gasse & cộng sự (2019).

- 6.3. Scaling. Scaling to larger problems can be a challenge. If a model trained on instances up to some size, say TSP up to size 50 nodes, is evaluated on larger instances, say TSP of size 100, 500 nodes, etc. challenge exists in terms of generalization, as mentioned in Sect. 4.3. Indeed, all of papers tackling TSP through ML & attempting to solve larger instances see degrading performance as size increases much beyond sizes seen during training (Bello et al., 2017; Khalil et al., 2017a; Kool & Welling, 2018; Vinyals et al., 2015). To tackle this issue, one may try to learn on larger instances, but this may turn out to be a computational & generalization issue. Except for very simple ML models & strong assumptions about data distribution, impossible to know computational complexity & sample complexity, i.e., number of observations that learning requires, because one is unaware of exact problem one is trying to solve, i.e., true data generating distribution.

  – Việc mở rộng quy mô cho các bài toán lớn hơn có thể là 1 thách thức. Nếu 1 mô hình được huấn luyện trên các trường hợp có kích thước nhất định, chẳng hạn như TSP có kích thước lên đến 50 nút, được đánh giá trên các trường hợp lớn hơn, chẳng hạn như TSP có kích thước 100, 500 nút, v.v., thì thách thức tồn tại về mặt khái quát hóa, như đã đề cập trong Mục 4.3. Thật vậy, tất cả các bài báo giải quyết TSP thông qua ML & cố gắng giải quyết các trường hợp lớn hơn đều thấy hiệu suất giảm sút khi kích thước tăng lên nhiều so với kích thước được thấy trong quá trình huấn luyện (Bello & cộng sự, 2017; Khalil & cộng sự, 2017a; Kool & Welling, 2018; Vinyals & cộng sự, 2015). Để giải quyết vấn đề này, người ta có thể thử học trên các trường hợp lớn hơn, nhưng điều này có thể trở thành vấn đề & khái quát hóa về mặt tính toán. Ngoại trừ các mô hình ML rất đơn giản & các giả định mạnh về phân phối dữ liệu, không thể biết độ phức tạp tính toán & độ phức tạp mẫu, tức là số lượng quan sát mà quá trình học yêu cầu, vì người ta không biết chính xác vấn đề mà mình đang cố gắng giải quyết, tức là phân phối tạo dữ liệu thực.

- 6.4. Data generation. Collecting data (e.g. instances of optimization problems) is a subtle task. Larsen et al. (2018) claim: "*sampling from historical data is appropriate when attempting to mimic a behavior reflected in such data*". I.e., given an external process on which we observe instances of an optimization problem, can collect data to train some policy needed for optimization, & expect policy to generalize on future instances of this application. A practical example would be a business that frequently encounters optimization problems related to their activities, e.g. Montreal delivery company example used in introduction.

  – Tạo dữ liệu. Việc thu thập dữ liệu (ví dụ: các trường hợp bài toán tối ưu hóa) là 1 nhiệm vụ tinh tế. Larsen & cộng sự (2018) khẳng định: "*việc lấy mẫu từ dữ liệu lịch sử là phù hợp khi cố gắng mô phỏng hành vi được phản ánh trong dữ liệu đó*". Tức là, với 1 quy trình bên ngoài mà chúng ta quan sát các trường hợp bài toán tối ưu hóa, có thể thu thập dữ liệu để huấn luyện 1 số chính sách cần thiết cho việc tối ưu hóa, & kỳ vọng chính sách này sẽ được tổng quát hóa cho các trường hợp ứng dụng này trong tương lai. Một ví dụ thực tế là 1 doanh nghiệp thường xuyên gặp phải các vấn đề tối ưu hóa liên quan đến hoạt động của họ, ví dụ như ví dụ về công ty giao hàng ở Montreal được sử dụng trong phần giới thiệu.

  In other cases, i.e., when we are not targeting a specific application for which we would have historical data, how can we proactively train a policy for problems that we do not yet know of? As partially discussed in Sect. 4.3, 1st need to define to which family of instances we want to generalize over. E.g., might decide to learn a cutting plane selection policy for Euclidean TSPs. Even so, it remains a complex effort to generate problems that capture essence of real applications. Moreover, combinatorial optimization problems are high dimensional, highly structured, & troublesome to visualize. Sole exercise of generating graphs is already a complicated one! Topic has nonetheless received some interest. Smith-Miles & Bowly (2015) claim: confidence we can put in an algorithm "*depends on how carefully we select test instances*". but note however that too often, a new algorithm is claimed "*to be superior by showing that it outperforms previous approaches on a set of well-studied instances*". Authors propose a problem instance generating method that consists of: defining an instance feature space visualizing it in 2D (using dimensionality reduction techniques e.g. principal component analysis), & using an evolutionary algorithm to drive instance generation toward a pre-defined sub-space. Authors argue: method is successful if easy & hard instances can be easily separated in reduced instance space. Methodology is then fruitfully applied to graph-based problems, but would require redefining evolution primitives in order to be applied to other type of problems.

On contrary, Malitsky, Merschformann, O'Sullivan, & Tierney (2016) propose a method to generate problem instances from same probability distribution, in that case, one of "*industrial*" boolean satisfiability problem instances. Authors use a large neighborhood search, using destruction & reparation primitives, to search for new instances. Some instance features are computed to classify whether new instances fall under same cluster as target one.

– Trong các trường hợp khác, tức là khi chúng ta không nhắm mục tiêu vào 1 ứng dụng cụ thể mà chúng ta đã có dữ liệu lịch sử, làm thế nào chúng ta có thể chủ động huấn luyện 1 chính sách cho các vấn đề mà chúng ta chưa biết? Như đã thảo luận 1 phần trong Phần 4.3, trước tiên cần xác định chúng ta muốn khái quát hóa trên họ trường hợp nào. Ví dụ, có thể quyết định học 1 chính sách lựa chọn mặt cắt cho các TSP Euclid. Tuy nhiên, việc tạo ra các bài toán nắm bắt được bản chất của các ứng dụng thực tế vẫn là 1 nỗ lực phức tạp. Hơn nữa, các bài toán tối ưu hóa tổ hợp có nhiều chiều, có cấu trúc cao & & khó hình dung. Việc chỉ tập trung vào việc tạo đồ thị đã là 1 vấn đề phức tạp! Tuy nhiên, chủ đề này vẫn nhận được 1 số sự quan tâm. Smith-Miles & Bowly (2015) khẳng định: chúng ta có thể đặt niềm tin vào 1 thuật toán "*nó phụ thuộc vào mức độ cẩn thận khi chúng ta lựa chọn các trường hợp thử nghiệm*". Tuy nhiên, cần lưu ý rằng thuật toán mới thường được tuyên bố "*nó vượt trội hơn bằng cách chứng minh rằng nó hoạt động tốt hơn các phương pháp trước đó trên 1 tập hợp các trường hợp đã được nghiên cứu kỹ lưỡng*". Các tác giả đề xuất 1 phương pháp tạo ra các thể hiện vấn đề bao gồm: xác định 1 không gian đặc trưng thể hiện bằng cách trực quan hóa nó trong 2D (sử dụng các kỹ thuật giảm chiều, ví dụ: phân tích thành phần chính), & sử dụng thuật toán tiến hóa để hướng việc tạo ra các thể hiện đến 1 không gian con được xác định trước. Các tác giả lập luận: phương pháp này thành công nếu các thể hiện dễ & khó có thể dễ dàng tách biệt trong không gian thể hiện được rút gọn. Phương pháp này sau đó được áp dụng hiệu quả cho các vấn đề dựa trên đồ thị, nhưng sẽ yêu cầu xác định lại các nguyên hàm tiến hóa để có thể áp dụng cho các loại vấn đề khác. Ngược lại, Malitsky, Merschformann, O'Sullivan, & Tierney (2016) đề xuất 1 phương pháp để tạo ra các thể hiện vấn đề từ cùng 1 phân phối xác suất, trong trường hợp đó, 1 trong các thể hiện vấn đề thỏa mãn boolean "*industrial*". Các tác giả sử dụng tìm kiếm lân cận lớn, sử dụng các nguyên hàm hủy & sửa chữa, để tìm kiếm các thể hiện mới. Một số đặc điểm thể hiện được tính toán để phân loại xem các thể hiện mới có thuộc cùng cụm với mục tiêu hay không.

Deciding how to represent data is also not an easy task, but can have a dramatic impact on learning. E.g., how does one properly represent a branch-&-bound node, or even whole branch-&-bound tree? These representations need to be expressive enough for learning, but at same time, concise enough to be used frequently without excessive computations.

– Việc quyết định cách biểu diễn dữ liệu cũng không phải là 1 nhiệm vụ dễ dàng, nhưng có thể có tác động đáng kể đến việc học. Ví dụ, làm thế nào để biểu diễn đúng 1 nút nhánh, hay thậm chí toàn bộ cây nhánh? Những biểu diễn này cần đủ biểu cảm cho việc học, nhưng đồng thời cũng phải đủ ngắn gọn để sử dụng thường xuyên mà không cần tính toán quá mức.

- 7. Conclusion. Surveyed & highlighted how ML can be used to build combinatorial optimization algorithms that are partially learned. Have suggested: imitation learning alone can be valuable if policy learned is significantly faster to compute than original one provided by an expert, in this case a combinatorial optimization algorithm. On contrary, models trained with a reward signal have potential to outperform current policies, given enough training & a supervised initialization. Training a policy that generalizes to unseen problems is a challenge, this is why we believe learning should occur on a distribution small enough that policy could fully exploit structure of problem & give better results. Believe end-to-end ML approaches to combinatorial optimization can be improved by using ML in combination with current combinatorial optimization algorithms to benefit from theoretical guarantees & state-of-art algorithms already available.

  – Đã khảo sát & nhấn mạnh cách ML có thể được sử dụng để xây dựng các thuật toán tối ưu hóa tổ hợp được học 1 phần. Đã đề xuất: chỉ riêng việc học bắt chước có thể có giá trị nếu chính sách được học nhanh hơn đáng kể so với chính sách gốc do chuyên gia cung cấp, trong trường hợp này là 1 thuật toán tối ưu hóa tổ hợp. Ngược lại, các mô hình được đào tạo bằng tín hiệu phần thưởng có tiềm năng vượt trội hơn các chính sách hiện tại, nếu được đào tạo đủ & khởi tạo có giám sát. Đào tạo 1 chính sách tổng quát hóa cho các vấn đề chưa thấy là 1 thách thức, đây là lý do tại sao chúng tôi tin rằng việc học nên diễn ra trên 1 phân phối đủ nhỏ để chính sách có thể khai thác hoàn toàn cấu trúc của vấn đề & mang lại kết quả tốt hơn. Tin rằng các phương pháp tiếp cận ML đầu cuối đối với tối ưu hóa tổ hợp có thể được cải thiện bằng cách sử dụng ML kết hợp với các thuật toán tối ưu hóa tổ hợp hiện tại để hưởng lợi từ các đảm bảo lý thuyết & các thuật toán tiên tiến hiện có.

  Other than performance incentives, there is also interest in using ML as a modeling tool for discrete optimization, as done by Lombardi & Milano (2018), or to extract intuition & knowledge about algorithms as mentioned in Bonami et al. (2018); Khalil et al. (2017a).

  – Ngoài các động cơ thúc đẩy hiệu suất, người ta cũng quan tâm đến việc sử dụng ML như 1 công cụ mô hình hóa để tối ưu hóa rời rạc, như Lombardi & Milano (2018) đã thực hiện, hoặc để trích xuất trực giác & kiến thức về thuật toán như đã đề cập trong Bonami et al. (2018); Khalil et al. (2017a).

  Although most of approaches discussed in this paper are still at an exploratory level of deployment, at least in terms of their use in general-purpose (commercial) solvers, strongly believe: this is just beginning of a new era for combinatorial optimization algorithms.

  – Mặc dù hầu hết các phương pháp được thảo luận trong bài báo này vẫn đang ở mức độ triển khai thăm dò, ít nhất là về mặt sử dụng trong các trình giải quyết mục đích chung (thương mại), nhưng chúng tôi tin chắc rằng: đây chỉ là khởi đầu của 1 kỷ nguyên mới cho các thuật toán tối ưu hóa tổ hợp.

3 Wikipedia's

3.1 Wikipedia/combinatorial optimization

Combinatorial optimization is a subfield of mathematical optimization that consists of finding an optimal object from a finite set of objects, where set of feasible solutions is discrete or can be reduced to a discrete set. Typical combinatorial optimization problems are traveling salesman problem (TSP), minimum spanning tree problem (MST), & knapsack problem. In many such problems, e.g. ones previously mentioned, exhaustive search is not tractable, & so specialized algorithms that quickly rule out large parts of search space or approximation algorithms must be resorted to instead.

– Tối ưu hóa tổ hợp là 1 nhánh của tối ưu hóa toán học, bao gồm việc tìm kiếm 1 đối tượng tối ưu từ 1 tập hợp hữu hạn các đối tượng, trong đó tập hợp các nghiệm khả thi là rời rạc hoặc có thể được rút gọn thành 1 tập rời rạc. Các bài toán tối ưu hóa tổ hợp điển hình bao gồm bài toán người bán hàng du lịch (TSP), bài toán cây bao trùm nhỏ nhất (MST), & bài toán ba lô. Trong nhiều bài toán như vậy, ví dụ như các bài toán đã đề cập trước đó, tìm kiếm vét cạn không khả thi, do đó, các thuật toán chuyên biệt loại trừ nhanh chóng các phần lớn không gian tìm kiếm hoặc các thuật toán xấp xỉ phải được sử dụng thay thế.

Fig: A minimum spanning tree of a weighted planar graph. Finding a minimum spanning tree is a common problem involving combinatorial optimization.

Combinatorial optimization is related to operations research, algorithm theory, & computational complexity theory. It has important applications in several fields, including AI, ML, auction theory, software engineering, VLSI, applied mathematics & theoretical CS.

– Tối ưu hóa tổ hợp liên quan đến nghiên cứu hoạt động, lý thuyết thuật toán & lý thuyết độ phức tạp tính toán. Nó có những ứng dụng quan trọng trong nhiều lĩnh vực, bao gồm AI, ML, lý thuyết đấu giá, kỹ thuật phần mềm, VLSI, toán ứng dụng & khoa học máy tính lý thuyết.

3.1.1 Applications

Basic applications of combinatorial optimization include, but are not limited to: logistics, supply chain optimization, developing best airline network of spoles & destinations, deciding which taxis in a fleet to route to pick up fares, determining optimal way to deliver packages, allocating jobs to people optimally, designing water distribution networks, earth science problems (e.g. reservoir flow-rates).

– Các ứng dụng cơ bản của tối ưu hóa tổ hợp bao gồm nhưng không giới hạn ở: hậu cần, tối ưu hóa chuỗi cung ứng, phát triển mạng lưới hàng không tốt nhất về điểm đến & điểm đến, quyết định tuyến taxi nào trong đội bay để đón khách, xác định cách tối ưu để giao hàng, phân bổ công việc cho mọi người 1 cách tối ưu, thiết kế mạng lưới phân phối nước, các vấn đề khoa học về trái đất (ví dụ: lưu lượng dòng chảy của hồ chứa).

3.1.2 Methods

There is a large amount of literature on polynomial-time algorithms for certain special classes of discrete optimization. A considerable amount of it is unified by theory of linear programming. Some examples of combinatorial optimization problems that are covered by this framework are shortest paths & shortest-path trees, flows & circulations, spanning trees, matching, & matroid problems.

– Có rất nhiều tài liệu về thuật toán thời gian đa thức cho 1 số lớp tối ưu hóa rời rạc đặc biệt. Phần lớn trong số đó được thống nhất bởi lý thuyết quy hoạch tuyến tính. 1 số ví dụ về các bài toán tối ưu hóa tổ hợp được đề cập trong khuôn khổ này bao gồm đường đi ngắn nhất & cây đường đi ngắn nhất, luồng & tuần hoàn, cây khung, bài toán ghép nối & matroid.

For NP-complete discrete optimization problems, current research literature includes following topics:

- polynomial-time exactly solvable special cases of problem at hand (e.g., fixed-parameter tractable problems)
- algorithms that perform well on "random" instances (e.g., for TSP)
- approximation algorithms that run in polynomial time & find a solution that is close to optimal
- solving real-world instances that arise in practice & do not necessarily exhibit worst-case behavior of in NP-complete problems (e.g. real-world TSP instances with 10s of thousands of nodes).

– Đối với các bài toán tối ưu rời rạc NP-đầy đủ, các tài liệu nghiên cứu hiện tại bao gồm các chủ đề sau:

- các trường hợp đặc biệt có thể giải chính xác trong thời gian đa thức của bài toán đang xét (ví dụ: các bài toán có thể giải được với tham số cố định)
- các thuật toán hoạt động tốt trên các trường hợp "ngẫu nhiên" (ví dụ: đối với TSP)
- các thuật toán xấp xỉ chạy trong thời gian đa thức & tìm ra giải pháp gần tối ưu
- giải các trường hợp thực tế phát sinh trong thực tế & không nhất thiết thể hiện hành vi xấu nhất của các bài toán NP-đầy đủ (ví dụ: các trường hợp TSP thực tế với hàng chục nghìn nút).

Combinatorial optimization problems can be viewed as searching for best element of some set of discrete items; therefore, in principle, any sort of search algorithm or metaheuristic can be used to solve them. Widely applicable approaches include branch-&-bound (an exact algorithm which can be stopped at any point in time to serve as heuristic), branch-&-cut (uses linear optimization to generate bounds), dynamic programming (a recursive solution construction with limited search window) & tabu search (a greedy-type swapping algorithm). However, generic search algorithms are not guaranteed to find an optimal solution 1st, nor are they guaranteed to run quickly (in polynomial time). Since some discrete optimization problems are NP-complete, e.g. traveling salesman (decision) problem, this is expected unless P = NP.

– Các bài toán tối ưu hóa tổ hợp có thể được xem như việc tìm kiếm phần tử tốt nhất của 1 tập hợp các phần tử rời rạc; do đó, về nguyên tắc, bất kỳ loại thuật toán tìm kiếm hoặc siêu thuật toán nào cũng có thể được sử dụng để giải chúng. Các phương pháp được áp dụng rộng rãi bao gồm branch-&-bound (một thuật toán chính xác có thể dừng tại bất kỳ thời điểm nào để làm thuật toán tìm kiếm), branch-&-cut (sử dụng tối ưu hóa tuyến tính để tạo ra các giới hạn), lập trình động (một cấu trúc giải pháp đệ quy với cửa sổ tìm kiếm giới hạn) & tìm kiếm & tabu (một thuật toán hoán đổi kiểu tham lam). Tuy nhiên, các thuật toán tìm kiếm chung không đảm bảo sẽ tìm ra giải pháp tối ưu đầu tiên, cũng như không đảm bảo chúng chạy nhanh (trong thời gian đa thức). Vì 1 số bài toán tối ưu hóa rời rạc là NP-đầy đủ, ví dụ như bài toán người bán hàng du lịch (quyết định), điều này là bình thường trừ khi P = NP.

For each combinatorial optimization problem, there is a corresponding decision problem that asks whether there is a feasible solution for some particular measure m_0. E.g., if there is a graph G which contains vertices u, v, an optimization problem might be "find a path from u to v that uses fewest edges". This problem might have an answer of, say, 4. A A corresponding decision problem would be "is there a path from u to v that uses 10 or fewer edges?" This problem can be answered with a simple yes or no.

– Với mỗi bài toán tối ưu tổ hợp, có 1 bài toán quyết định tương ứng đặt ra câu hỏi liệu có 1 giải pháp khả thi cho 1 số đo m_0 cụ thể nào đó hay không. Ví dụ, nếu có 1 đồ thị G chứa các đỉnh u, v, 1 bài toán tối ưu hóa có thể là "tìm đường đi từ u đến v sử dụng ít cạnh nhất". Bài toán này có thể có đáp án là, chẳng hạn, 4. 1 bài toán quyết định tương ứng sẽ là "có đường đi từ u đến v sử dụng 10 cạnh trở xuống không?" Bài toán này có thể được trả lời đơn giản bằng có hoặc không.

Field of approximation algorithms deals with algorithms to find near-optimal solutions to hard problems. Usual decision version is then an inadequate definition of problem since it only specifies acceptable solutions. Even though could introduce suitable decision problems, problem is then more naturally characterized as an optimization problem.

– Lĩnh vực thuật toán xấp xỉ tập trung vào các thuật toán tìm kiếm các giải pháp gần tối ưu cho các bài toán khó. Phiên bản quyết định thông thường khi đó là 1 định nghĩa không đầy đủ về bài toán vì nó chỉ xác định các giải pháp chấp nhận được. Mặc dù có thể đưa ra các bài toán quyết định phù hợp, nhưng bài toán khi đó được mô tả 1 cách tự nhiên hơn là bài toán tối ưu hóa.

3.1.3 NP optimization problem

An *NP-optimization problem* (NPO) is a combinatorial optimization problem with following additional conditions. Note: below referred polynomials are functions of size of respective functions' inputs, not size of some implicit set of input instances.

- size of every feasible solution $y \in f(x)$ is polynomially bounded in size of given instance x,
- languages $\{x; x \in I\}$ & $\{(x, y); y \in f(x)\}$ can be recognized in polynomial time, &
- m is polynomial-time computable.

This implies: corresponding decision problem is in NP. In computer science, interesting optimization problems usually have above properties & are therefore NPO problems. A problem is additionally called a P-optimization (PO) problem, if there exists an algorithm which finds optimal solution in polynomial time. Often, when dealing with class NPO, one is interested in optimization problems for which decision versions are NP-complete. Note: hardness relations are always w.r.t. some reduction. Due to connection between approximation algorithms & computational optimization problems, reductions which preserve approximation in some respect are for this subject preferred than usual Turing & Karp reductions. An example of such a reduction would be L-reduction. For this reason, optimization problems with NP-complete decision versions are not necessarily called NPO-complete.

– Bài toán tối ưu hóa NP (NPO) là 1 bài toán tối ưu hóa tổ hợp với các điều kiện bổ sung sau. Lưu ý: các đa thức được đề cập dưới đây là các hàm phụ thuộc vào kích thước của các đầu vào tương ứng, chứ không phải kích thước của 1 tập hợp các thể hiện đầu vào ngầm định.

- Kích thước của mọi nghiệm khả thi $y \in f(x)$ bị chặn đa thức theo kích thước của thể hiện x đã cho,
- Các ngôn ngữ $\{x; x \in I\}$ & $\{(x, y); y \in f(x)\}$ có thể được nhận dạng trong thời gian đa thức, &
- m có thể tính toán được trong thời gian đa thức.

Điều này ngụ ý: bài toán quyết định tương ứng nằm trong NP. Trong khoa học máy tính, các bài toán tối ưu hóa thú vị thường có các tính chất trên & do đó là các bài toán NPO. 1 bài toán còn được gọi là bài toán tối ưu hóa P (PO) nếu tồn tại 1 thuật toán tìm ra nghiệm tối ưu trong thời gian đa thức. Thông thường, khi xử lý lớp NPO, người ta quan tâm đến các bài toán tối ưu hóa mà các phiên bản quyết định là NP-đầy đủ. Lưu ý: các quan hệ độ cứng luôn liên quan đến 1 số phép rút gọn. Do mối liên hệ giữa các thuật toán xấp xỉ & các bài toán tối ưu hóa tính toán, các phép rút gọn bảo toàn xấp xỉ ở 1 số khía cạnh được ưu tiên hơn so với các phép rút gọn Turing & Karp thông thường. 1 ví dụ về phép rút gọn như vậy là phép rút gọn L. Vì lý do này, các bài toán tối ưu hóa với các phiên bản quyết định NP-đầy đủ không nhất thiết được gọi là NPO-đầy đủ.

NPO is divided into following subclasses according to their approximability:

- NPO(I): Equals FPTAS. Contains Knapsack problem.
- NPO(II): Equals PTAS. Contains Makespan scheduling problem.
- NPO(III): Class of NPO problems that have polynomial-time algorithms which computes solution with a cost at most c times optimal cost (for minimization problems) or a cost at least $\frac{1}{c}$ of optimal cost (for maximization problems). In Hromkovič's book, excluded from this class are all NPO(II)-problems save if P = NP. Without exclusion, equals APX. Contains MAX-SAT & metric TSP.
- NPO(IV): Class of NPO problems with polynomial-time algorithms approximating optimal solution by a ratio that is polynomial in a logarithm of size of input. In Hromkovič's book, all NPO(III)-problems are excluded from this class unless P = NP. Contains set cover problem.
- NPO(V): Class of NPO problems with polynomial-time algorithms approximating optimal solution by a ratio bounded by some function on n. In Hromkovic's book, all NPO(IV)-problems are excluded from this class unless P = NP. Contains TSP & clique problem.

– NPO được chia thành các lớp con sau theo tính xấp xỉ của chúng:

- NPO(I): Bằng FPTAS. Chứa bài toán Knapsack.
- NPO(II): Bằng PTAS. Chứa bài toán lập lịch Makespan.
- NPO(III): Lớp các bài toán NPO có thuật toán thời gian đa thức tính toán nghiệm với chi phí tối đa là c lần chi phí tối ưu (đối với bài toán cực tiểu hóa) hoặc chi phí tối thiểu là $\frac{1}{c}$ chi phí tối ưu (đối với bài toán cực đại hóa). Trong sách của Hromkovič, tất cả các bài toán NPO(II) trừ khi P = NP đều bị loại trừ khỏi lớp này. Nếu không loại trừ, bằng APX. Chứa MAX-SAT & metric TSP.
- NPO(IV): Lớp các bài toán NPO với thuật toán thời gian đa thức tính toán nghiệm tối ưu theo tỷ lệ đa thức trong logarit kích thước đầu vào. Trong sách của Hromkovič, tất cả các bài toán NPO(III) đều bị loại khỏi lớp này trừ khi P = NP. Chứa bài toán phủ tập hợp.
- NPO(V): Lớp các bài toán NPO với thuật toán thời gian đa thức xấp xỉ nghiệm tối ưu theo tỷ lệ bị chặn bởi 1 hàm số nào đó trên n. Trong sách của Hromkovic, tất cả các bài toán NPO(IV) đều bị loại khỏi lớp này trừ khi P = NP. Chứa bài toán TSP & clique.

An NPO problem is called polynomially bounded (PB) if, for every instance x & for every solution $y \in f(x)$, measure $m(x, y)$ is bounded by a polynomial function of size of x. Class NPOPB is class NPO problems that are polynomially-bounded.

– 1 bài toán NPO được gọi là bị chặn đa thức (PB) nếu, với mọi trường hợp x & với mọi nghiệm $y \in f(x)$, độ đo $m(x, y)$ bị chặn bởi 1 hàm đa thức có độ lớn x. Lớp NPOPB là lớp các bài toán NPO bị chặn đa thức.

3.1.4 Specific problems

Assignment problem, bin packing problem, Chinese postman problem, closure problem, constraint satisfaction problem, cutting stock problem, dominating set problem, integer programming, job shop scheduling, knapsack problem, metric k-center/vertex k-center problem, minimum relevant variables in linear system, minimum spanning tree, nurse scheduling problem, ring star problem, set cover problem, talent scheduling, traveling salesman problem, vehicle rescheduling problem, vehicle routing problem, weapon target assignment problem.

– Bài toán gán, bài toán đóng thùng, bài toán người đưa thư Trung Quốc, bài toán đóng, bài toán thỏa mãn ràng buộc, bài toán cắt kho, bài toán tập hợp trội, lập trình số nguyên, lập lịch xưởng gia công, bài toán ba lô, bài toán tâm đỉnh k, biến liên quan tối thiểu trong hệ thống tuyến tính, cây bao trùm tối thiểu, bài toán lập lịch y tá, bài toán ngôi sao vòng, bài toán che phủ tập hợp, lập lịch tài năng, bài toán nhân viên bán hàng du lịch, bài toán lập lịch lại phương tiện, bài toán định tuyến phương tiện, bài toán chỉ định mục tiêu vũ khí.

4 Miscellaneous

Tài liệu

[KV18] Bernhard Korte and Jens Vygen. *Combinatorial optimization*. Vol. 21. Algorithms and Combinatorics. Theory and algorithms, Sixth edition of [MR1764207]. Springer, Berlin, 2018, pp. xxi+698. ISBN: 978-3-662-56038-9; 978-3-662-56039-6. DOI: 10.1007/978-3-662-56039-6. URL: https://doi.org/10.1007/978-3-662-56039-6.