Survey: Artificial Intelligence – Khảo Sát: Trí Tuệ Nhân Tạo

Nguyễn Quản Bá Hồng*

Ngày 19 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:

- *Survey: Artificial Intelligence – Khảo Sát: Trí Tuệ Nhân Tạo.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/NQBH_AI.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/NQBH_AI.tex.
- *Lecture Note: Introduction to Artificial Intelligence – Bài Giảng: Nhập Môn Trí Tuệ Nhân Tạo.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/lecture/NQBH_introduction_AI_lecture.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/lecture/NQBH_introduction_AI_lecture.tex.
- Codes:
  - C++: https://github.com/NQBH/advanced_STEM_beyond/tree/main/AI/C++.
  - Python: https://github.com/NQBH/advanced_STEM_beyond/tree/main/AI/Python.

Mục lục

1 Basic AI

1.1 [NR21]. Peter Norvig, Stuart Russell. Artificial Intelligence: A Modern Approach

[479 Amazon ratings][4365 Goodreads ratings]

- Amazon review. The long-anticipated revision of *AI: A Modern Approach* explores full breadth & depth of field of AI. 4e brings readers up to date on latest technologies, presents concepts in a more unified manner, & offers new or expanded coverage of ML, DL, transfer learning, multi agent systems, robotics, NLP, causality, probabilistic programming, privacy, fairness, & safe AI.
- Preface. AI is a big field, & this is a big book. Have tried to explore full breadth of field, which encompasses logic, probability, & continuous mathematics; perception, reasoning, learning, & actions; fairness, trust, social good, & safety; & applications that range from microelectronic devices to robotic planetary explorers to online services with billions of users.

  Subtitle of this book is "A Modern Approach". I.e., have chosen to tell story from a current perspective. Synthesize what is now known into a common framework, recasting early work using ideas & terminology that are prevalent today. Apologize to those whose subfields are, as a result, less recognizable.

*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.

- New to 4e. This edition reflects changes of AI since last edition in 2010:
  - Focus more on ML rather than hand-crafted knowledge engineering, due to increased availability of data, computing resources, & new algorithms.
  - DL, probabilistic programming, & multiagent systems receive expanded coverage, each with their own chap.
  - Coverage of natural language understanding, robotics, & computer vision has been revised to reflect impact of DL.
  - Robotics chap now includes robots that interact with humans & application of reinforcement learning to robotics.
  - Previously defined goal of AI as creating systems that try to maximize expected utility, where specific utility information – objective – is supplied by human designers of system. Now we no longer assume: objective is fixed & known by AI system; instead, system may be uncertain about true objectives of humans on whose behalf it operates. It must learn what to maximize & must function appropriately even while uncertain about objective.
  - Increase coverage of impact of AI on society, including vital issues of ethics, fairness, trust, & safety.
  - Have moved exercises from end of each chap to an online site. This allows us to continuously add to, update, & improve exercises, to meet needs of instructors & to reflect advances in field & in AI-related software tools.
  - Overall, about 25% of material in book is brand new. Remaining 75% has been largely rewritten to present a more unified picture of field. 22% of citations of 4e are to works published after 2010.
- Overview of book. Main unifying theme is idea of an *intelligent agent*. Define AI as study of agents that receive percepts from environment & perform actions. Each such agent implements a function that maps percept sequences to actions, & cover different ways to represent these functions, e.g. reactive agents, real-time planners, decision-theoretic systems, & DL systems. Emphasize learning both as a construction method for competent systems & as a way of extending reach of designer into unknown environments. Treat robotics & vision not as independently defined problems, but as occurring in service of achieving goals. Stress importance of task environment in determining appropriate agent design.

  – Chủ đề thống nhất chính là ý tưởng về 1 *tác nhân thông minh*. Định nghĩa AI là nghiên cứu về các tác nhân tiếp nhận các nhận thức từ môi trường & thực hiện các hành động. Mỗi tác nhân như vậy triển khai 1 chức năng ánh xạ các chuỗi nhận thức thành các hành động, & bao gồm các cách khác nhau để biểu diễn các chức năng này, ví dụ như các tác nhân phản ứng, các nhà lập kế hoạch thời gian thực, các hệ thống lý thuyết quyết định, & hệ thống DL. Nhấn mạnh việc học vừa là phương pháp xây dựng cho các hệ thống có năng lực & như 1 cách mở rộng phạm vi của nhà thiết kế vào các môi trường chưa biết. Xử lý robot & tầm nhìn không phải là các vấn đề được xác định độc lập, mà là diễn ra để phục vụ cho việc đạt được các mục tiêu. Nhấn mạnh tầm quan trọng của môi trường nhiệm vụ trong việc xác định thiết kế tác nhân phù hợp.

  Primary aim: convey *ideas* that have emerged over past 70 years of AI research & past 2 millennia of related work. Have tried to avoid excessive formality in presentation of these ideas, while retaining precision. Have included mathematical formulas & pseudocode algorithms to make key ideas concrete; mathematical concepts & notation are described in Appendix A & our pseudocode is described in Appendix B.

  – Mục tiêu chính: truyền đạt *ý tưởng* đã xuất hiện trong hơn 70 năm nghiên cứu AI & 2 thiên niên kỷ công trình liên quan. Đã cố gắng tránh sự trang trọng quá mức trong việc trình bày những ý tưởng này, đồng thời vẫn giữ được độ chính xác. Đã bao gồm các công thức toán học & thuật toán mã giả để làm cho các ý tưởng chính trở nên cụ thể; các khái niệm toán học & ký hiệu được mô tả trong Phụ lục A & mã giả của chúng tôi được mô tả trong Phụ lục B.

  This book is primarily intended for use in an undergraduate course or course sequence. Book has 29 chaps, each requiring about a week's worth of lectures, so working through whole book requires a 2-semester sequence. 1 1-semester course can use selected chaps to suit interests of instructor & students. Book can also be used in a graduate-level course (perhaps with addition of some of primary courses suggested in bibliographical notes), or for self-study or as a reference.

  Only prerequisite is familiarity with basic concepts of CS (algorithms, data structures, complexity) at a sophomore level. Freshman calculus & linear algebra are useful for some of topics.

PART I: AI.

- 1. Introduction. In which we try to explain why we consider AI to be a subject most worthy of study, & in which we try to decide what exactly it is, this being a good thing to decide before embarking.

  Call ourselves *Homo sapiens* – man the wise – because our *intelligence* is so important to us. For thousands of years, have tried to understand *how we think & act* – i.e., how our brain, a mere handful of matter, can perceive, understand, predict, & manipulate a world far larger & more complicated than itself. Field of AI is concerned with not just understanding but also *building* intelligent entities – machines that can compute how to act effectively & safely in a wide variety of novel situations.

  Surveys regularly rank AI as 1 of most interesting & fastest-growing fields, & already generating over a trillion dollars a year in revenue. AI expert Kai-Fu Lee predicts: its impact will be "more than anything in history of mankind". Moreover, intellectual frontiers of AI are wide open. Whereas a student of an older science e.g. physics might feel best ideas have already been discovered by Galileo, Newton, Curie, Einstein, & the rest, AI still has many openings for full-time masterminds.

  AI currently encompasses a huge variety of subfields, ranging from general (learning, reasoning, perception, etc.) to specific, e.g. playing chess, proving mathematical theorems, writing poetry, driving a car, or diagnosing diseases. AI is relevant to any intellectual task; it is truly a universal field.

  - 1.1. What Is AI? Have claimed: AI is interesting, but have not said what it is. Historically, researchers have pursued several different versions of AI. Some have defined intelligence in terms of fidelity to *human* performance, while others

prefer an abstract, formal def of intelligence called *rationality* – loosely speaking, doing "right thing". Subject matter itself also varies: some consider intelligence to be a property of internal *thought processes & reasoning*, while others focus on intelligent *behavior*, an external characterization. [In public eye, there is sometimes confusion between terms "AI" & "ML". ML is a subfield of AI that studies ability to improve performance based on experience. Some AI systems use ML methods to achieve competence, but some do not.]
– Đã tuyên bố: AI rất thú vị, nhưng chưa nói rõ nó là gì. Theo truyền thống, các nhà nghiên cứu đã theo đuổi 1 số phiên bản khác nhau của AI. 1 số người đã định nghĩa trí thông minh theo nghĩa là sự trung thành với hiệu suất của *con người*, trong khi những người khác thích 1 định nghĩa trừu tượng, chính thức về trí thông minh được gọi là *lý trí* – nói 1 cách rộng rãi, làm "điều đúng đắn". Bản thân chủ đề cũng khác nhau: 1 số coi trí thông minh là 1 đặc tính của *quá trình suy nghĩ & lý luận* bên trong, trong khi những người khác tập trung vào *hành vi* thông minh, 1 đặc điểm bên ngoài. [Trong mắt công chúng, đôi khi có sự nhầm lẫn giữa các thuật ngữ "AI" & "ML". ML là 1 lĩnh vực phụ của AI nghiên cứu khả năng cải thiện hiệu suất dựa trên kinh nghiệm. 1 số hệ thống AI sử dụng các phương pháp ML để đạt được năng lực, nhưng 1 số thì không.]
From these 2 dimensions – human vs. rational [We are not suggesting that humans are "irrational" in dictionary sense of "deprived of normal mental clarity". We are merely conceding that human decisions are not always mathematically perfect.] & thought vs. behavior – there are 4 possible combinations, & there have been adherents & research programs ∀ 4. Methods used are necessarily different: pursuit of human-like intelligence must be in part an empirical science related to psychology, involving observations & hypotheses about actual human behavior & thought processes; a rationalist approach, on other hand, involves a combination of mathematics & engineering, & connects to statistics, control theory, & economics. Various groups have both disparaged & helped each other. Look at 4 approaches in more detail.
– Từ 2 chiều này – con người so với lý trí [Chúng tôi không ám chỉ rằng con người "phi lý trí" theo nghĩa trong từ điển là "thiếu sự minh mẫn bình thường về mặt tinh thần". Chúng tôi chỉ thừa nhận rằng các quyết định của con người không phải lúc nào cũng hoàn hảo về mặt toán học.] & suy nghĩ so với hành vi – có 4 sự kết hợp có thể, & đã có những người ủng hộ & các chương trình nghiên cứu ∀ 4. Các phương pháp được sử dụng nhất thiết phải khác nhau: việc theo đuổi trí thông minh giống con người phải là 1 phần khoa học thực nghiệm liên quan đến tâm lý học, bao gồm các quan sát & giả thuyết về hành vi thực tế của con người & các quá trình suy nghĩ; mặt khác, 1 cách tiếp cận duy lý bao gồm sự kết hợp của toán học & kỹ thuật, & kết nối với thống kê, lý thuyết kiểm soát, & kinh tế học. Nhiều nhóm đã coi thường & giúp đỡ lẫn nhau. Hãy xem xét 4 cách tiếp cận chi tiết hơn.

· 1.1.1. Acting humanly: Turing test approach. *Turing test*, proposed by Alan Turing (1950) was designed as a thought experiment that would sidestep philosophical vagueness of question "Can a machine think?" A computer passes test if a human interrogator, after posing some written questions, cannot tell whether written responses come from a person or from a computer. Chap. 28 discusses details of test & whether a computer would really be intelligent if it passed. For now, note: programming a computer to pass a rigorously applied test provides plenty to work on. Computer would need following capabilities:

1. *natural language processing* to communicate successfully in a human language
2. *knowledge representation* to store what it knows or hears
3. *automated reasoning* to answer questions & to draw new conclusions
4. *ML* to adapt to new circumstances & to detect & extrapolate patterns.

Turing viewed *physical* simulation of a person as unnecessary to demonstrate intelligence. However, other researchers have proposed a *total Turing test*, which requires interaction with objects & people in real world. To pass total Turing test, a robot will need

1. *computer vision* & speech recognition to perceive world
2. *robotics* to manipulate objects & move about.

These 6 disciplines compose most of AI. Yet AI researchers have devoted little effort to passing Turing test, believing: more important to study underlying principles of intelligence. Quest for "artificial flight" succeeded when engineers & investors stopped imitating birds & started using wind tunnels & learning about aerodynamics. Aeronautical engineering texts do not define goal of their fields as making "machines that fly so exactly like pigeons that they can fool even other pigeons".
– 6 chuyên ngành này tạo nên phần lớn AI. Tuy nhiên, các nhà nghiên cứu AI đã dành ít nỗ lực để vượt qua bài kiểm tra Turing, tin rằng: quan trọng hơn là nghiên cứu các nguyên tắc cơ bản của trí thông minh. Nhiệm vụ tìm kiếm "chuyến bay nhân tạo" đã thành công khi các kỹ sư & nhà đầu tư ngừng bắt chước chim & bắt đầu sử dụng đường hầm gió & tìm hiểu về khí động học. Các văn bản về kỹ thuật hàng không không xác định mục tiêu của lĩnh vực này là tạo ra "những cỗ máy bay giống hệt chim bồ câu đến mức chúng có thể đánh lừa cả những con bồ câu khác".

· 1.1.2. Thinking humanly: cognitive modeling approach. To say a program thinks like a human, must know how humans think. Can learn about human though in 3 ways:

1. *introspection* – trying to catch our own thoughts as they go by
2. *psychological experiments* – observing a person in action
3. *brain imaging* – observing brain in action.

Once have a sufficiently precise theory of mind, it becomes possible to express theory as a computer program. If program's input–output behavior matches corresponding human behavior, that is evidence: some of program's mechanisms could also be operating in humans.

– 1 khi có 1 lý thuyết đủ chính xác về tâm trí, có thể diễn đạt lý thuyết như 1 chương trình máy tính. Nếu hành vi đầu vào-đầu ra của chương trình khớp với hành vi tương ứng của con người, thì đó là bằng chứng: 1 số cơ chế của chương trình cũng có thể hoạt động ở con người.

E.g., Allen Newell & Herbert Simon, who developed GPS, "General Problem Solver" (Newell & Simon, 1961), were not content merely to have their program solve problems correctly. They were more concerned with comparing sequence & timing of its reasoning steps to those of human subjects solving same problems. Interdisciplinary field of *cognitive science* brings together computer models from AI & experimental techniques from psychology to construct precise & testable theories of human mind.

– Ví dụ, Allen Newell & Herbert Simon, người đã phát triển GPS, "General Problem Solver" (Newell & Simon, 1961), không chỉ hài lòng với việc chương trình của họ giải quyết vấn đề 1 cách chính xác. Họ quan tâm nhiều hơn đến việc so sánh trình tự & thời gian của các bước lý luận của nó với trình tự của con người giải quyết cùng 1 vấn đề. Lĩnh vực liên ngành của *khoa học nhận thức* tập hợp các mô hình máy tính từ AI & các kỹ thuật thử nghiệm từ tâm lý học để xây dựng các lý thuyết chính xác & có thể kiểm chứng về tâm trí con người.

Cognitive science is a fascinating field in itself, worthy of several textbooks & at least 1 encyclopedia (Wilson & Keil, 1999). Will occasionally comment on similarities or differences between AI techniques & human cognition. Real cognition science, however, is necessary based on experimental investigation of actual humans or animals. Leave that for other books, as assume reader has only a computer for experimentation.

– Khoa học nhận thức là 1 lĩnh vực hấp dẫn, xứng đáng với 1 số sách giáo khoa & ít nhất 1 bách khoa toàn thư (Wilson & Keil, 1999). Thỉnh thoảng sẽ bình luận về điểm tương đồng hoặc khác biệt giữa các kỹ thuật AI & nhận thức của con người. Tuy nhiên, khoa học nhận thức thực sự là cần thiết dựa trên nghiên cứu thực nghiệm trên con người hoặc động vật thực tế. Hãy để dành điều đó cho các cuốn sách khác, vì giả sử người đọc chỉ có máy tính để thử nghiệm.

In early days of AI there was often confusion between approaches. An author would argue: an algorithm performs well on a task & therefor a good model of human performance, or vice versa. Modern authors separate 2 kinds of claims; this distinction has allowed both AI & cognitive science to develop more rapidly. 2 fields fertilize each other, most notably in computer vision, which incorporates neurophysiological evidence into computational models. Recently, combination of neuroimaging methods combined with ML techniques for analyzing such data has led to beginnings of a capability to "read minds" – i.e., to ascertain semantic content of a person's inner thoughts. This capability could, in turn, shed further light on how human cognitive works.

– Vào những ngày đầu của AI, thường có sự nhầm lẫn giữa các cách tiếp cận. 1 tác giả sẽ lập luận: 1 thuật toán thực hiện tốt 1 nhiệm vụ & do đó là 1 mô hình tốt về hiệu suất của con người, hoặc ngược lại. Các tác giả hiện đại tách biệt 2 loại tuyên bố; sự phân biệt này đã cho phép cả AI & khoa học nhận thức phát triển nhanh hơn. 2 lĩnh vực này hỗ trợ lẫn nhau, đáng chú ý nhất là trong lĩnh vực thị giác máy tính, nơi kết hợp bằng chứng thần kinh sinh lý vào các mô hình tính toán. Gần đây, sự kết hợp của các phương pháp chụp ảnh thần kinh kết hợp với các kỹ thuật ML để phân tích dữ liệu như vậy đã dẫn đến sự khởi đầu của khả năng "đọc suy nghĩ" – tức là xác định nội dung ngữ nghĩa của những suy nghĩ bên trong của 1 người. Đến lượt mình, khả năng này có thể làm sáng tỏ thêm cách thức hoạt động của nhận thức của con người.

· 1.1.3. Thinking rationally: "law of thought" approach. Greek philosopher Aristotle was 1 of 1st attempt to codify "right thinking" – i.e., irrefutable reasoning processes. His *syllogisms* provided patterns for argument structures that always yielded correct conclusions when given correct premises. Canonical example starts with *Socrates is a man & all men are moral* & concludes *Socrates is mortal*. (This example is probably due to Sextus Empiricus rather than Aristotle). These laws of thought were supposed to govern operation of mind; their study initiated field called *logic*.

Logicians in 19th century developed a precise notation for statements about objects in world & relations among them. (Contrast this with ordinary arithmetic notation, which provides only for statements about *numbers*.) By 1965, programs could, in principle, solve *any* solvable problem described in logical notation. So-called *logicist* tradition within AI hopes to build on such programs to create intelligent systems.

– Các nhà logic học vào thế kỷ 19 đã phát triển 1 ký hiệu chính xác cho các phát biểu về các đối tượng trong thế giới & các mối quan hệ giữa chúng. (Đối chiếu điều này với ký hiệu số học thông thường, chỉ cung cấp các phát biểu về *số*.) Đến năm 1965, về nguyên tắc, các chương trình có thể giải quyết *bất kỳ* vấn đề có thể giải quyết nào được mô tả bằng ký hiệu logic. Cái gọi là truyền thống logic trong AI hy vọng sẽ xây dựng trên các chương trình như vậy để tạo ra các hệ thống thông minh.

Logic as conventionally understood requires knowledge of world that is *certain* – a condition that, in reality, is seldom achieved. Simply don't know rules of, say, politics or warfare in same way that we know rules of chess or arithmetic. Theory of *probability* fills this gap, allowing rigorous reasoning with uncertain information. In principle, it allows construction of a comprehensive model of rational thought, leading from raw perceptual information to an understanding of how world works to predictions about future. What it does not do, is generate intelligent *behavior*. For that, we need a theory of rational action. Rational thought, by itself, is not enough.

– Logic theo cách hiểu thông thường đòi hỏi kiến thức về thế giới *chắc chắn* – 1 điều kiện mà trên thực tế hiếm khi đạt được. Đơn giản là không biết các quy tắc của, chẳng hạn, chính trị hay chiến tranh theo cùng cách mà chúng ta biết các quy tắc của cờ vua hay số học. Lý thuyết về *xác suất* lấp đầy khoảng trống này, cho phép lý luận chặt chẽ với thông tin không chắc chắn. Về nguyên tắc, nó cho phép xây dựng 1 mô hình toàn diện về tư duy hợp lý, dẫn từ thông tin nhận thức thô sơ đến sự hiểu biết về cách thế giới hoạt động để dự đoán về tương lai. Điều mà nó không làm được là tạo ra *hành vi* thông minh. Đối với điều đó, chúng ta cần 1 lý thuyết về hành động hợp lý. Tư duy hợp lý, tự nó, là không đủ.

· 1.1.4. Acting rationally: rational agent approach. An *agent* is juts sth that acts (*agent* comes from Latin *agere*, to do). Of

course, all computer programs do sth, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, & create & pursue goals. A *rational agent* is one that acts so as to achieve best outcome or, when there is uncertainty, best expected outcome.
– *Hành động hợp lý: phương pháp tiếp cận tác nhân hợp lý.* 1 tác nhân chỉ là cái gì đó hành động (tác nhân bắt nguồn từ tiếng Latin agere, nghĩa là làm). Tất nhiên, tất cả các chương trình máy tính đều làm cái gì đó, nhưng các tác nhân máy tính được kỳ vọng sẽ làm nhiều hơn thế: hoạt động tự chủ, nhận thức môi trường của chúng, tồn tại trong 1 khoảng thời gian dài, thích nghi với sự thay đổi, & tạo ra & theo đuổi mục tiêu. 1 tác nhân hợp lý là tác nhân hành động để đạt được kết quả tốt nhất hoặc, khi có sự không chắc chắn, kết quả mong đợi tốt nhất.
In "laws of thought" approach to AI, emphasis was on correct inferences. Making correct inferences is sometimes *part* of being a rational agent, because 1 way to act rationally: deduce that a given action is best & then to act on that conclusion. On other hand, there are ways of acting rationally that cannot be said to involve inference. E.g., recoiling from a hot stove is a reflex action that is usually more successful than a slower action taken after careful deliberation.
– Trong cách tiếp cận "luật tư duy" đối với AI, trọng tâm là suy luận đúng. Việc đưa ra suy luận đúng đôi khi là *1 phần* của việc trở thành 1 tác nhân lý trí, bởi vì 1 cách để hành động hợp lý: suy ra rằng 1 hành động nhất định là tốt nhất & sau đó hành động theo kết luận đó. Mặt khác, có những cách hành động hợp lý không thể nói là liên quan đến suy luận. Ví dụ, lùi lại khỏi bếp nóng là 1 hành động phản xạ thường thành công hơn so với hành động chậm hơn được thực hiện sau khi cân nhắc kỹ lưỡng.
All skills needed for Turing test also allow an agent to act rationally. Knowledge representation & reasoning enable agents to reach good decisions. Need to be able to generate comprehensible sentences in natural language to get by in a complex society. Need learning not only for erudition, but also because it improves our ability to generate effective behavior, especially in circumstances that are new.
– Tất cả các kỹ năng cần thiết cho bài kiểm tra Turing cũng cho phép 1 tác nhân hành động hợp lý. Biểu diễn kiến thức & lý luận cho phép các tác nhân đưa ra quyết định đúng đắn. Cần có khả năng tạo ra các câu dễ hiểu bằng ngôn ngữ tự nhiên để tồn tại trong 1 xã hội phức tạp. Cần học không chỉ để có kiến thức uyên bác mà còn vì nó cải thiện khả năng tạo ra hành vi hiệu quả của chúng ta, đặc biệt là trong những hoàn cảnh mới.
Rational-agent approach to AI has 2 advantages over other approaches. 1st, it is more general than "law of thought" approach because correct inference is just 1 of several possible mechanism for achieving rationality. 2nd, more amenable to scientific development. Standard of rationality is mathematically well defined & completely general. Can often work back from this specification to derive agent designs that provably achieve it – sth that is largely impossible if goal: imitate human behavior or thought processes.
– Phương pháp tiếp cận tác nhân hợp lý đối với AI có 2 ưu điểm so với các phương pháp tiếp cận khác. Thứ nhất, nó tổng quát hơn phương pháp tiếp cận "luật tư duy" vì suy luận đúng chỉ là 1 trong số nhiều cơ chế có thể đạt được tính hợp lý. Thứ hai, dễ tiếp thu hơn đối với sự phát triển khoa học. Tiêu chuẩn của tính hợp lý được định nghĩa rõ ràng về mặt toán học & hoàn toàn tổng quát. Thường có thể làm việc ngược lại từ thông số kỹ thuật này để đưa ra các thiết kế tác nhân có thể chứng minh được là đạt được nó – điều mà phần lớn là không thể nếu mục tiêu: bắt chước hành vi hoặc quá trình suy nghĩ của con người.
For these reasons, rational-agent approach to AI has prevailed throughout most of field's history. In early decades, rational agents were built on logical foundations & formed definite plans to achieve specific goals. Later, methods based on probability theory & ML allowed creation of agents that could make decisions under uncertainty to attain best expected outcome. In a nutshell, *AI has focused on study & construction of agents that do right thing.* What counts as right thing is defined by objective that we provide to agent. This general paradigm is so pervasive that we might call it *standard model.* It prevails not only in AI, but also in control theory, where a controller minimizes a cost function; in operations research, where a policy maximizes a sum of rewards; in statistics, where a decision rule minimizes a loss function; & in economics, where a decision maker maximizes utility or some measure of social welfare.
– Vì những lý do này, cách tiếp cận tác nhân hợp lý đối với AI đã chiếm ưu thế trong hầu hết lịch sử của lĩnh vực này. Trong những thập kỷ đầu, các tác nhân hợp lý được xây dựng trên nền tảng logic & hình thành các kế hoạch chắc chắn để đạt được các mục tiêu cụ thể. Sau đó, các phương pháp dựa trên lý thuyết xác suất & ML cho phép tạo ra các tác nhân có thể đưa ra quyết định trong điều kiện không chắc chắn để đạt được kết quả mong đợi tốt nhất. Tóm lại, *AI đã tập trung vào nghiên cứu & xây dựng các tác nhân làm điều đúng đắn.* Những gì được coi là điều đúng đắn được xác định bởi mục tiêu mà chúng ta cung cấp cho tác nhân. Mô hình chung này rất phổ biến đến mức chúng ta có thể gọi nó là *mô hình chuẩn.* Nó không chỉ chiếm ưu thế trong AI mà còn trong lý thuyết điều khiển, trong đó bộ điều khiển giảm thiểu hàm chi phí; trong nghiên cứu hoạt động, trong đó chính sách tối đa hóa tổng phần thưởng; trong thống kê, trong đó quy tắc quyết định giảm thiểu hàm mất mát; & trong kinh tế, trong đó người ra quyết định tối đa hóa tiện ích hoặc 1 số biện pháp phúc lợi xã hội.
Need to make 1 important refinement to standard model to account for fact that perfect rationality – always taking exactly optimal action – is not feasible in complex environments. Computational demands are just too high. Chaps. 6 & 16 deals with issue of *limited rationality* – acting appropriately when there is not enough time to do all computations one might like. However, perfect rationality often remains a good starting point for theoretical analysis.
– Cần thực hiện 1 cải tiến quan trọng đối với mô hình chuẩn để tính đến thực tế là tính hợp lý hoàn hảo – luôn thực hiện hành động tối ưu chính xác – là không khả thi trong các môi trường phức tạp. Yêu cầu tính toán quá cao. Chương 6 & 16 giải quyết vấn đề về *tính hợp lý hạn chế* – hành động phù hợp khi không có đủ thời gian để thực hiện tất cả các phép tính mà người ta có thể thích. Tuy nhiên, tính hợp lý hoàn hảo thường vẫn là điểm khởi đầu tốt cho phân tích lý thuyết.

· 1.1.5. Beneficial machines. Standard model has been a useful guide for AI research since its inception, but it is probably not right model in long run. Reason: standard model assumes: will supply a fully specified objective to machine.
– Mô hình chuẩn đã là 1 hướng dẫn hữu ích cho nghiên cứu AI kể từ khi ra đời, nhưng có lẽ về lâu dài, nó không phải là mô hình phù hợp. Lý do: mô hình chuẩn giả định: sẽ cung cấp 1 mục tiêu được chỉ định đầy đủ cho máy.
For an artificially defined task e.g. chess or shortest-path computation, task comes with an objective built in – so standard model is applicable. As move into real world, however, it becomes more & more difficult to specify objective completely & correctly. E.g., in designing a self-driving car, one might think: objective is to reach destination safely. But driving along any road incurs a risk of injury due to other errant drivers, equipment failure, etc.; thus, a strict goal of safety requires staying in garage. There is a tradeoff between making progress towards destination & incurring a risk of injury. How should this tradeoff be made? Furthermore, to what extent can we allow car to take actions that would annoy other drivers? How much should car moderate its acceleration, steering, & braking to avoid shaking up passenger? These kinds of questions are difficult to answer a priori. They are particularly problematic in general area of human–robot interaction, of which self-driving car is 1 example.
– Đối với 1 nhiệm vụ được xác định nhân tạo, ví dụ như cờ vua hoặc tính toán đường đi ngắn nhất, nhiệm vụ đi kèm với 1 mục tiêu được tích hợp sẵn – do đó, mô hình chuẩn có thể áp dụng được. Tuy nhiên, khi chuyển sang thế giới thực, việc xác định mục tiêu 1 cách hoàn chỉnh & chính xác trở nên & khó khăn hơn. Ví dụ, khi thiết kế 1 chiếc xe tự lái, người ta có thể nghĩ: mục tiêu là đến đích an toàn. Nhưng việc lái xe trên bất kỳ con đường nào cũng có nguy cơ bị thương do những người lái xe khác đi sai đường, hỏng thiết bị, v.v.; do đó, mục tiêu an toàn nghiêm ngặt đòi hỏi phải ở trong gara. Có 1 sự đánh đổi giữa việc tiến về đích & với nguy cơ bị thương. Sự đánh đổi này nên được thực hiện như thế nào? Hơn nữa, chúng ta có thể cho phép xe thực hiện những hành động có thể làm phiền những người lái xe khác ở mức độ nào? Xe nên điều chỉnh gia tốc, đánh lái, & phanh ở mức nào để tránh làm rung chuyển hành khách? Những loại câu hỏi này rất khó trả lời trước. Chúng đặc biệt có vấn đề trong lĩnh vực tương tác giữa con người & rô-bốt nói chung, trong đó xe tự lái là 1 ví dụ.
Problem of achieving agreement between our true preferences & objective we put into machine is called *value alignment problem*: values or objectives put into machine must be aligned with those of human. In we are developing an AI system in lab or in a simulator – as has been case for most of field's history – there is an easy fix for an incorrectly specified objective: reset system, fix objective, & try again. As field progresses towards increasingly capable intelligent systems that are deployed in rewal world, this approach is no longer viable. A system deployed with an incorrect objective will have negative consequences. Moreover, more intelligent system, more negative consequences.
– Vấn đề đạt được sự đồng thuận giữa sở thích thực sự của chúng ta & mục tiêu mà chúng ta đưa vào máy được gọi là vấn đề căn chỉnh giá trị: các giá trị hoặc mục tiêu đưa vào máy phải phù hợp với mục tiêu hoặc mục tiêu của con người. Khi chúng ta đang phát triển 1 hệ thống AI trong phòng thí nghiệm hoặc trong trình mô phỏng – như đã từng xảy ra trong hầu hết lịch sử của lĩnh vực này – có 1 cách khắc phục dễ dàng cho 1 mục tiêu được chỉ định không chính xác: đặt lại hệ thống, sửa mục tiêu, & thử lại. Khi lĩnh vực này tiến triển theo hướng các hệ thống thông minh ngày càng có khả năng được triển khai trong thế giới mới, cách tiếp cận này không còn khả thi nữa. 1 hệ thống được triển khai với mục tiêu không chính xác sẽ có hậu quả tiêu cực. Hơn nữa, hệ thống càng thông minh thì hậu quả tiêu cực càng nhiều.
Returning to apparently unproblematic example of chess, consider what happens if machine is intelligent enough to reason & act beyond confines of chessboard. In that case, it might attempt to increase its chances of winning by such ruses as hypnotizing or blackmailing its opponent or bribing audience to make rustling noises during its opponent's thinking time. [In 1 of 1st books on chess, Ruy Lopez (1561) wrote, "Always place board so sun is in your opponent's eyes."] It might also attempt to hijack additional computing power for itself. *These behaviors are not "unintelligent" or "insane"; they are a logical consequence of defining winning as the sole objective for machine.*
– Quay trở lại ví dụ cờ vua có vẻ không có vấn đề gì, hãy xem xét điều gì xảy ra nếu máy đủ thông minh để lý luận & hành động vượt ra ngoài giới hạn của bàn cờ. Trong trường hợp đó, nó có thể cố gắng tăng cơ hội chiến thắng bằng những mánh khóe như thôi miên hoặc tống tiền đối thủ hoặc hối lộ khán giả tạo ra tiếng sột soạt trong thời gian suy nghĩ của đối thủ. [Trong 1 trong những cuốn sách đầu tiên về cờ vua, Ruy Lopez (1561) đã viết, "Luôn đặt bàn cờ sao cho mặt trời chiếu vào mắt đối thủ."] Nó cũng có thể cố gắng chiếm đoạt thêm sức mạnh tính toán cho chính nó. *Những hành vi này không phải là "không thông minh" hay "điên rồ"; chúng là hệ quả hợp lý của việc xác định chiến thắng là mục tiêu duy nhất của máy.*
Impossible to anticipate all ways in which a machine pursuing a fixed objective might misbehave. There is good reason, then, to think: standard model is inadequate. We don't want machines that are intelligent in sense of pursuing *their* objectives; want them to pursue *our* objectives. If cannot transfer those objectives perfectly to machine, then need a new formulation – one in which machine is pursuing our objectives, but is necessarily *uncertain* as to what they are. When a machine knows that it doesn't know complete objective, it has an incentive to act cautiously, to ask permission, to learn more about our preferences through observation, & to defer to human control. Ultimately, want agents that are *provably beneficial* to humans.
– Không thể lường trước được mọi cách mà 1 cỗ máy theo đuổi 1 mục tiêu cố định có thể hoạt động không đúng. Do đó, có lý do chính đáng để nghĩ rằng: mô hình chuẩn là không đủ. Chúng ta không muốn những cỗ máy thông minh theo nghĩa theo đuổi *là mục tiêu của chúng*; muốn chúng theo đuổi *là mục tiêu của chúng ta*. Nếu không thể chuyển những mục tiêu đó 1 cách hoàn hảo cho cỗ máy, thì cần 1 công thức mới – công thức mà trong đó cỗ máy đang theo đuổi mục tiêu của chúng ta, nhưng nhất thiết *không chắc chắn* về mục tiêu đó là gì. Khi 1 cỗ máy biết rằng nó không biết mục tiêu hoàn chỉnh, nó có động cơ để hành động thận trọng, để xin phép, để tìm hiểu thêm về sở thích của chúng ta thông qua quan sát, & để tuân theo sự kiểm soát của con người. Cuối cùng, muốn các tác nhân *có thể chứng minh được là có lợi* cho con người.

∗ 1.2. Foundations of AI. In this sect, provide a brief history of disciplines that contributed ideas, viewpoints, & techniques to AI. Like any history, this one concentrates on a small number of people, events, & ideas & ignores others that also were important. Organize history around a series of questions. Certainly would not wish to give impression that these questions are only ones the disciplines address or disciplines have all been working toward AI as their ultimate fruition.
– *Nền tảng của AI.* Trong phần này, hãy cung cấp 1 lịch sử tóm tắt về các ngành đã đóng góp ý tưởng, quan điểm, & kỹ thuật cho AI. Giống như bất kỳ lịch sử nào, phần này tập trung vào 1 số ít người, sự kiện, & ý tưởng & bỏ qua những người khác cũng quan trọng. Sắp xếp lịch sử xung quanh 1 loạt các câu hỏi. Chắc chắn không muốn tạo ấn tượng rằng đây chỉ là những câu hỏi mà các ngành giải quyết hoặc tất cả các ngành đều hướng tới AI như là thành quả cuối cùng của họ.

· 1.2.1. Philosophy.

1. Can formal rules be used to draw valid conclusions?
2. How does mind arise from a physical brain?
3. Where does knowledge come from?
4. How does knowledge lead to action?

p. 24+++

○ 2. Intelligent Agents. In which we discuss nature of agents, perfect or otherwise, diversity of environments, & resulting menagerie of agent types.

– *Các tác nhân thông minh.* Trong đó chúng ta thảo luận về bản chất của các tác nhân, hoàn hảo hay không, sự đa dạng của môi trường, & sự kết hợp của các loại tác nhân.

Chap. 1 identified concept of *rational agents* as central as our approach to AI. In this chap, make this notion more concrete. See: concept of rationality can be applied to a wide variety of agents operating in any imaginable environment. Our plan in this book: use this concept to develop a small set of design principles for building successful agents – systems that can reasonably be called *intelligent*.

– Chương 1 xác định khái niệm về các tác nhân hợp lý là trung tâm trong cách tiếp cận của chúng ta đối với AI. Trong chương này, hãy cụ thể hóa khái niệm này hơn. Xem: khái niệm về tính hợp lý có thể được áp dụng cho nhiều loại tác nhân hoạt động trong bất kỳ môi trường nào có thể tưởng tượng được. Kế hoạch của chúng tôi trong cuốn sách này: sử dụng khái niệm này để phát triển 1 tập hợp nhỏ các nguyên tắc thiết kế nhằm xây dựng các tác nhân thành công – các hệ thống có thể được gọi 1 cách hợp lý là thông minh.

Begin by examining agents, environments, & coupling between them. Observation that some agents behave better than others leads naturally to idea of a rational agent – one that behaves as well as possible. How well an agent can behave depends on nature of environment; some environments are more difficult than others. Give a crude categorization of environments & show how properties of an environment influence design of suitable agents for that environment. Describe a number of basic "skeleton" agent designs, which flesh out in rest of book.

– Bắt đầu bằng cách xem xét các tác nhân, môi trường, & sự kết hợp giữa chúng. Quan sát thấy 1 số tác nhân hành xử tốt hơn những tác nhân khác dẫn đến ý tưởng về 1 tác nhân hợp lý – 1 tác nhân hành xử tốt nhất có thể. Mức độ 1 tác nhân có thể hành xử tốt như thế nào phụ thuộc vào bản chất của môi trường; 1 số môi trường khó hơn những môi trường khác. Đưa ra 1 phân loại thô sơ về các môi trường & cho thấy các đặc tính của 1 môi trường ảnh hưởng đến thiết kế các tác nhân phù hợp cho môi trường đó như thế nào. Mô tả 1 số thiết kế tác nhân "bộ xương" cơ bản, được trình bày chi tiết trong phần còn lại của cuốn sách.

∗ 2.1. Agents & Environments. An *agent* is anything that can be viewed as perceiving its *environment* through *sensors* & acting upon that environment through *actuators*. This simple idea is illustrated in Fig. 2.1: Agents interact with environments through sensors & actuators. A human agent has eyes, ears, & other organs for sensors & hands, legs, vocal tract, & so on for actuators. A robotic agent might have cameras & infrared range finders for sensors & various motors for actuators. A software agent receives file contents, network packets, & human input (keyboard/mouse/touchscreen/voice) as sensory inputs & acts on environment by writing files, sending network packets, & displaying information or generating sounds. Environment could be everything – entire universe! In practice it is just that part of universe whose state we care about when designing this agent – part that affects what agent perceives & is affected by agent's actions.
– *Tác nhân & Môi trường.* 1 tác nhân là bất kỳ thứ gì có thể được xem như nhận thức môi trường của nó thông qua các cảm biến & tác động lên môi trường đó thông qua các bộ truyền động. Ý tưởng đơn giản này được minh họa trong Hình 2.1: Các tác nhân tương tác với môi trường thông qua các cảm biến & bộ truyền động. 1 tác nhân con người có mắt, tai, & các cơ quan khác để cảm biến & tay, chân, đường thanh quản, & v.v. để truyền động. 1 tác nhân rô bốt có thể có camera & máy đo khoảng cách hồng ngoại để cảm biến & nhiều động cơ khác nhau để truyền động. 1 tác nhân phần mềm nhận nội dung tệp, các gói mạng, & đầu vào của con người (bàn phím/chuột/màn hình cảm ứng/giọng nói) làm đầu vào cảm biến & tác động lên môi trường bằng cách ghi tệp, gửi các gói mạng, & hiển thị thông tin hoặc tạo ra âm thanh. Môi trường có thể là tất cả mọi thứ – toàn bộ vũ trụ! Trên thực tế, chỉ có 1 phần của vũ trụ mà chúng ta quan tâm đến trạng thái của nó khi thiết kế tác nhân này – phần ảnh hưởng đến những gì tác nhân nhận thức & bị ảnh hưởng bởi các hành động của tác nhân.
Use term *percept* to refer to content an agent's sensors are perceiving. An agent's *percept sequence* is complete history of everything agent has ever perceived. In general, *an agent's choice of action at any given instant can depend on its built-in knowledge & on entire percept sequence observed to date, but not on anything it hasn't perceived.* By specifying agent's choice of action for every possible percept sequence, have said more or less everything there is to say about agent.

Mathematically speaking, say: an agent's behavior is described by *agent function* that maps any given percept sequence to an action.

– Sử dụng thuật ngữ percept để chỉ nội dung mà các cảm biến của tác nhân đang nhận thức. Chuỗi nhận thức của tác nhân là lịch sử hoàn chỉnh về mọi thứ mà tác nhân từng nhận thức. Nhìn chung, *lựa chọn hành động của tác nhân tại bất kỳ thời điểm nào có thể phụ thuộc vào kiến thức tích hợp của tác nhân & vào toàn bộ chuỗi nhận thức được quan sát cho đến nay, nhưng không phụ thuộc vào bất kỳ điều gì mà tác nhân chưa nhận thức.* Bằng cách chỉ định lựa chọn hành động của tác nhân cho mọi chuỗi nhận thức có thể, đã nói ít nhiều mọi thứ cần nói về tác nhân. Về mặt toán học, nói: hành vi của tác nhân được mô tả bởi *hàm tác nhân* ánh xạ bất kỳ chuỗi nhận thức nào cho 1 hành động.

Can imagine *tabulating* agent function that describes any given agent; for most agents, this would be a very large table – infinite, in fact, unless we place a bound on length of percept sequences we want to consider. Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences & recording which actions agent does in response. [If agent uses some randomization to choose its actions, then would have to try each sequence many times to identify probability of each action. One might imagine: acting randomly is rather silly, but show later in this chap: it can be very intelligent.] Table is, of course, an *external* characterization of agent. *Internally*, agent function for an artificial agent will be implemented by an *agent program*. Important to keep these 2 ideas distinct. Agent function is an abstract mathematical description; agent program is a concrete implementation, running within some physical system.

– Có thể tưởng tượng việc lập bảng hàm tác nhân mô tả bất kỳ tác nhân nào; đối với hầu hết các tác nhân, đây sẽ là 1 bảng rất lớn – vô hạn, trên thực tế, trừ khi chúng ta đặt 1 giới hạn về độ dài của các chuỗi nhận thức mà chúng ta muốn xem xét. Với 1 tác nhân để thử nghiệm, về nguyên tắc, chúng ta có thể xây dựng bảng này bằng cách thử tất cả các chuỗi nhận thức có thể & ghi lại hành động mà tác nhân thực hiện để phản hồi. [Nếu tác nhân sử dụng 1 số ngẫu nhiên để chọn hành động của mình, thì sẽ phải thử từng chuỗi nhiều lần để xác định xác suất của từng hành động. Người ta có thể tưởng tượng: hành động ngẫu nhiên khá ngớ ngẩn, nhưng sẽ hiển thị sau trong chương này: nó có thể rất thông minh.] Tất nhiên, bảng là 1 đặc điểm *bên ngoài* của tác nhân. *Bên trong*, hàm tác nhân cho 1 tác nhân nhân tạo sẽ được triển khai bởi 1 *chương trình tác nhân*. Điều quan trọng là phải giữ cho 2 ý tưởng này riêng biệt. Hàm tác nhân là 1 mô tả toán học trừu tượng; chương trình tác nhân là 1 triển khai cụ thể, chạy trong 1 số hệ thống vật lý.

To illustrate these ideas, use a simple example – vacuum-cleaner world, which consists of a robotic vacuum-cleaning agent in a world consisting squares that can be either dirty or clean. Fig. 2.2: A vacuum-cleaner world with just 2 locations. Each location can be clean or dirty, & agent can move left or right & can clean square that it occupies. Different versions of vacuum world allow for different rules about what agent can perceive, whether its actions always succeed, & so on. shows a configuration with just 2 squares, A & B. Vacuum agent perceives which square it is in & whether there is dirt in square. Agent starts in square A. Available actions are to move to right, move to left, suck up dirt, or do nothing. [In a real robot, it would be unlikely to have an actions like "move right" & "move left". Instead actions would be "spin wheels forward" & "spin wheels backward". Have chosen actions to be easier to follow on page, not for ease of implementation in an actual robot.] 1 very simple agent function is following: if current square is dirty, then suck; otherwise, move to other square. A partial tabulation of this agent function is shown in Fig. 2.3: Partial tabulation of a simple agent function for vacuum-cleaner world shown in Fig. 2.2. Agent cleans current square if it is dirty, otherwise it moves to other square. Note: table is of unbounded size unless there is a restriction on length of possible percept sequences. & an agent program that implements it appears in Fig. 2.8: Agent program for a simple reflex agent in 2-location vacuum environment. This program implements agent function tabulated in Fig. 2.3.

Looking at Fig. 2.3, see: various vacuum-world agents can be defined simply by filling in RH column in various ways. Obvious question, then: *What is right way to fill out table?* I.e., what makes an agent good or bad, intelligent or stupid? Answer these questions in next sect.

Before close this sect, should emphasize: notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides world into agents & non-agents. One could view a hand-held calculator as an agent that chooses action of displaying "4" when given percept sequence "$2 + 2 =$", but such an analysis would hardly aid our understanding of calculator. In a sense, all areas of engineering can be seen as designing artifacts that interact with world; AI operates at (what authors consider to be) most interesting end of spectrum, where artifacts have significant computational resources & task environment requires nontrivial decision making.

– Trước khi kết thúc phần này, cần nhấn mạnh: khái niệm về tác nhân được hiểu là 1 công cụ để phân tích hệ thống, không phải là 1 đặc điểm tuyệt đối chia thế giới thành tác nhân & không phải tác nhân. Người ta có thể xem máy tính cầm tay như 1 tác nhân chọn hành động hiển thị "4" khi đưa ra chuỗi nhận thức "$2 + 2 =$", nhưng 1 phân tích như vậy khó có thể giúp chúng ta hiểu máy tính. Theo 1 nghĩa nào đó, tất cả các lĩnh vực kỹ thuật đều có thể được coi là thiết kế các hiện vật tương tác với thế giới; AI hoạt động ở (những gì các tác giả coi là) phần cuối của quang phổ thú vị nhất, nơi các hiện vật có tài nguyên tính toán đáng kể & môi trường tác vụ đòi hỏi phải đưa ra quyết định không tầm thường.

∗ 2.2. Good Behavior: Concept of Rationality. A *rational agent* is one that does right thing. Obviously, doing right thing is better than doing wrong thing, but what does it mean to do right thing?

· 2.2.1. Performance measures. Moral philosophy has developed several different notions of "right thing", but AI has generally stuck to 1 notion called *consequentialism*: evaluate an agent's behavior by its consequences. When an agent is plunked down in an environment, it generates a sequence of actions according to percepts it receives. This sequence of actions causes environment to go through a sequence of states. If sequence is desirable, then agent has performed well. This notion of desirability is captured by a *performance measure* that evaluates any given sequence of environment states.

– *Các biện pháp hiệu suất.* Triết học đạo đức đã phát triển 1 số khái niệm khác nhau về "điều đúng đắn", nhưng AI

thường gắn bó với 1 khái niệm gọi là chủ nghĩa hậu quả: đánh giá hành vi của tác nhân theo hậu quả của nó. Khi 1 tác nhân được đặt xuống trong 1 môi trường, nó sẽ tạo ra 1 chuỗi hành động theo các nhận thức mà nó nhận được. Chuỗi hành động này khiến môi trường trải qua 1 chuỗi trạng thái. Nếu chuỗi là mong muốn, thì tác nhân đã hoạt động tốt. Khái niệm mong muốn này được nắm bắt bằng 1 *biện pháp hiệu suất* đánh giá bất kỳ chuỗi trạng thái môi trường nào được đưa ra.

Humans have desires & preferences of their own, so notion of rationality as applied to humans has to do with their success in choosing actions that produce sequences of environment states that are desirable *from their point of view.* Machines, on other hand, do not have desires & preferences of their own; performance measure is, initially at least, in mind of designer of machine, or in mind of users machine is designed for. See: some agent designs have an explicit representation of (a version of) performance measure, while in other designs performance measure is entirely implicit – agent may do right thing, but it doesn't know why.

– Con người có ham muốn & sở thích riêng, vì vậy khái niệm về tính hợp lý khi áp dụng cho con người có liên quan đến thành công của họ trong việc lựa chọn các hành động tạo ra chuỗi trạng thái môi trường mong muốn *theo quan điểm của họ.* Mặt khác, máy móc không có ham muốn & sở thích riêng; thước đo hiệu suất, ít nhất là ban đầu, nằm trong tâm trí của nhà thiết kế máy móc hoặc trong tâm trí của người dùng mà máy móc được thiết kế cho. Xem: 1 số thiết kế tác nhân có biểu diễn rõ ràng về (một phiên bản) thước đo hiệu suất, trong khi trong các thiết kế khác, thước đo hiệu suất hoàn toàn ngầm định – tác nhân có thể làm đúng, nhưng không biết tại sao.

Recalling Norbert Wiener's warning to ensure "purpose put into machine is purpose which we really desire", notice: it can be quite hard to formulate a performance measure correctly. Consider, e.g., vacuum–cleaner agent from preceding sect. Might propose to measure performance by amount of dirt cleaned up in a single 8-hour shift. With a rational agent, of course, what you ask for is what you get. A rational agent can maximize this performance measure by cleaning up dirt, then dumping it all on floor, then cleaning it up again, & so on. A more suitable performance measure would reward agent for having a clean floor. E.g., 1 point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed & noise generated). *As a general rule, better to design performance measures according to what one actually wants to be achieved in environment, rather than according to how one thinks agent should behave.*

– Nhắc lại lời cảnh báo của Norbert Wiener để đảm bảo "mục đích đưa vào máy là mục đích mà chúng ta thực sự mong muốn", hãy lưu ý: có thể khá khó để xây dựng 1 thước đo hiệu suất chính xác. Ví dụ, hãy xem xét tác nhân máy hút bụi từ phần trước. Có thể đề xuất đo hiệu suất theo lượng bụi bẩn được làm sạch trong 1 ca làm việc 8 giờ. Tất nhiên, với 1 tác nhân hợp lý, những gì bạn yêu cầu là những gì bạn nhận được. 1 tác nhân hợp lý có thể tối đa hóa thước đo hiệu suất này bằng cách làm sạch bụi bẩn, sau đó đổ hết xuống sàn, rồi lại làm sạch, & cứ như vậy. 1 thước đo hiệu suất phù hợp hơn sẽ thưởng cho tác nhân vì có sàn sạch. Ví dụ, có thể thưởng 1 điểm cho mỗi ô vuông sạch tại mỗi bước thời gian (có thể kèm theo hình phạt cho lượng điện tiêu thụ & tiếng ồn tạo ra). *Theo nguyên tắc chung, tốt hơn là thiết kế các thước đo hiệu suất theo những gì người ta thực sự muốn đạt được trong môi trường, thay vì theo cách người ta nghĩ tác nhân nên hành xử.*

Even when obvious pitfalls are avoided, some knotty problems remain. E.g., notion of "clean floor" in preceding paragraph is based on average cleanliness over time. Yet same average cleanliness can be achieved by 2 different agents, one of which does a mediocre job all time while other cleans energetically but takes long breaks. Which is preferable might seem to be a fine point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications. Which is better – an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor? Leave these questions as an exercise for diligent reader.

– Ngay cả khi tránh được những cạm bẫy rõ ràng, 1 số vấn đề nan giải vẫn còn tồn tại. Ví dụ, khái niệm "sàn nhà sạch" trong đoạn trước dựa trên mức độ sạch trung bình theo thời gian. Tuy nhiên, mức độ sạch trung bình tương tự có thể đạt được bởi 2 tác nhân khác nhau, 1 trong số đó làm việc tầm thường mọi lúc trong khi tác nhân kia làm việc rất hăng hái nhưng lại nghỉ giải lao rất lâu. Cái nào tốt hơn có vẻ là 1 điểm tinh tế của khoa học vệ sinh, nhưng trên thực tế, đó là 1 câu hỏi triết học sâu sắc với những hàm ý sâu xa. Cái nào tốt hơn – 1 nền kinh tế mà mọi người đều sống trong cảnh nghèo đói vừa phải, hay 1 nền kinh tế mà 1 số người sống trong cảnh sung túc trong khi những người khác lại rất nghèo? Hãy để những câu hỏi này như 1 bài tập cho người đọc siêng năng.

For most of book, assume: performance measure can be specified correctly. For reasons given above, however, must accept possibility that we might put wrong purpose into machine – precisely King Midas problem described on p. 51. Moreover, when designing 1 piece of software, copies of which will belong to different users, cannot anticipate exact preferences of each individual user. Thus, may need to build agents that reflect initial uncertainty about true performance measure & learn more about it as time goes by; such agents are described in Chaps. 15, 17, 23.

· 2.2.2. Rationality. What is rational at any given time depends on 4 things:

1. Performance measure that defines criterion of success.
2. Agent's prior knowledge of environment.
3. Actions that agent can perform.
4. Agent's percept sequence to date.

This leads to a def of a rational agent:

Definition 1 (Rational agent)**.** *For each possible percept sequence, a* rational agent *should select an action that is expected to maximize its performance measure, given evidence provided by percept sequence & whatever built-in knowledge agent has.*

– Đối với mỗi chuỗi nhận thức có thể, 1 tác nhân hợp lý *nên chọn 1 hành động dự kiến sẽ tối đa hóa thước đo hiệu suất của nó, dựa trên bằng chứng được cung cấp bởi chuỗi nhận thức & bất kỳ tác nhân kiến thức tích hợp nào.*

Consider simple vacuum-cleaner agent that cleans a square if it is dirty & moves to other square if not; this is agent function tabulated in Fig. 2.3. Is this a rational agent? That depends! 1st, need to say what performance measure is, what is known about environment, & what sensors & actuators agent has. Assume:

1. Performance measure awards 1 point for each clean square at each time step, over a "lifetime" of 1000 time steps.
2. "Geography" of environment is known *a priori* Fig. 2.2 but dirt distribution & initial location of agent are not. Clean squares stay clean & sucking cleans current square. *Right & Left* actions move agent 1 square except when this would take agent outside environment, in which case agent remains where it is.
3. Only available actions are *Right, Left, & Suck*.
4. Agent correctly perceives its location & whether that location contains dirt.

Under these circumstances agent is indeed rational; its expected performance is at least as good as other agent's.
One can see easily: same agent would be irrational under different circumstances. E.g., once all dirt is cleaned up, agent will oscillate needlessly back & forth; if performance measure includes a penalty of 1 point for each movement, agent will fare poorly.tA better agent for this case would do nothing once it is sure: all squares are clean. If clean squares can become dirty again, agent should occasionally check & reclean them if needed. If geography of environment is unknown, agent will need to *explore* it. Exercise 2.VACR asks you to design agents for these cases.
– Người ta có thể dễ dàng thấy: cùng 1 tác nhân sẽ không hợp lý trong những hoàn cảnh khác nhau. Ví dụ, sau khi tất cả bụi bẩn được dọn sạch, tác nhân sẽ dao động qua lại không cần thiết &; nếu thước đo hiệu suất bao gồm hình phạt 1 điểm cho mỗi chuyển động, tác nhân sẽ hoạt động kém. 1 tác nhân tốt hơn cho trường hợp này sẽ không làm gì cả khi chắc chắn: tất cả các ô vuông đều sạch. Nếu các ô vuông sạch có thể bị bẩn trở lại, tác nhân nên thỉnh thoảng kiểm tra & làm sạch lại chúng nếu cần. Nếu địa lý của môi trường không xác định, tác nhân sẽ cần phải *khám phá* nó. Bài tập 2.VACR yêu cầu bạn thiết kế các tác nhân cho những trường hợp này.

· 2.2.3. Omniscience, learning, & autonomy. Need to be careful to distinguish between rationality & *omniscience*. An omniscient agent knows *actual* outcome of its actions & can act accordingly; but omniscience is impossible in reality. Consider example: I am walking along Champs Elysées 1 day & I see an old friend across street. There is no traffic nearby & I'm not otherwise engaged, so, being rational, I start to cross street. Meanwhile, at 33000 feet, a cargo door falls off a passing airliner [See N. HENDERSON, "New door latches urged for Boeing 747 jumbo jets," Washington Post, Aug 24, 1989.], & before I make it to other side of street I am flattened. Was I irrational to cross street? It is unlikely: my obituary would read "Idiot attempts to cross street."
This example shows: rationality is not same as perfection. Rationality maximizes *expected* performance, while perfection maximizes *actual* performance. Retreating from a requirement of perfection is not just a question of being fair to agents. Point: if expect an agent to do what turns out after fact to be best action, it will be impossible to design an agent to fulfill this specification – unless improve performance of crystal balls or time machines.
– Ví dụ này cho thấy: tính hợp lý không giống với sự hoàn hảo. Tính hợp lý tối đa hóa hiệu suất *mong đợi*, trong khi sự hoàn hảo tối đa hóa hiệu suất *thực tế*. Việc rút lui khỏi yêu cầu về sự hoàn hảo không chỉ là vấn đề công bằng với các tác nhân. Điểm chính: nếu mong đợi 1 tác nhân thực hiện những gì sau này trở thành hành động tốt nhất, thì sẽ không thể thiết kế 1 tác nhân để đáp ứng thông số kỹ thuật này – trừ khi cải thiện hiệu suất của quả cầu pha lê hoặc cỗ máy thời gian.
Our definition of rationality does not require omniscience, then, because rational choice depends only on percept sequence *to date*. Must also ensure: we haven't inadvertently allowed agent to engage in decidedly underintelligent activities. E.g., if an agent does not look both ways before crossing a busy road, then its percept sequence will not tell it that there is a large truck approaching at high speed. Does our definition of rationality say that it's now OK to cross road? Far from it!
– Định nghĩa của chúng ta về tính hợp lý không đòi hỏi sự toàn năng, vì sự lựa chọn hợp lý chỉ phụ thuộc vào chuỗi nhận thức *cho đến nay*. Cũng phải đảm bảo: chúng ta không vô tình cho phép tác nhân tham gia vào các hoạt động rõ ràng là thiếu thông minh. Ví dụ, nếu 1 tác nhân không nhìn cả hai hướng trước khi băng qua 1 con đường đông đúc, thì chuỗi nhận thức của tác nhân đó sẽ không cho tác nhân biết rằng có 1 chiếc xe tải lớn đang lao tới với tốc độ cao. Định nghĩa của chúng ta về tính hợp lý có nói rằng bây giờ có thể băng qua đường không? Hoàn toàn không phải vậy! 1st, it would not be rational to cross road given this uninformative percept sequence: risk of accident from crossing without looking is too great. 2nd, a rational agent should choose "looking" action before stepping into street, because looking helps maximize expected performance. Doing actions *in order to modify future percepts* – sometimes called *information gathering* – is an important part of rationality & is covered in depth in Chap. 15. A 2nd example of information gathering is provided by *exploration* that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.
– 1. Sẽ không hợp lý khi băng qua đường khi xét đến chuỗi nhận thức không cung cấp thông tin này: nguy cơ tai nạn khi băng qua đường mà không nhìn là quá lớn. 2. 1 tác nhân hợp lý nên chọn hành động "nhìn" trước khi bước vào đường, vì nhìn giúp tối đa hóa hiệu suất mong đợi. Thực hiện các hành động *để sửa đổi các nhận thức trong tương lai* – đôi khi được gọi là *thu thập thông tin* – là 1 phần quan trọng của tính hợp lý & được trình bày sâu trong Chương 15. Ví dụ thứ 2 về việc thu thập thông tin được cung cấp bởi *khám phá* phải được thực hiện bởi 1 tác nhân hút bụi trong 1 môi trường ban đầu không xác định.
Our definition requires a rational agent not only to gather information but also to *learn* as much as possible from what it perceives. Agent's initial configuration could reflect some prior knowledge of environment, but as agent gains experience this may be modified & augmented. There are extreme cases in which environment is completely known *a priori* & completely predictable. In such cases, agent need not perceive or learn; it simply acts correctly.

– Định nghĩa của chúng tôi yêu cầu 1 tác nhân lý trí không chỉ thu thập thông tin mà còn *học* càng nhiều càng tốt từ những gì nó nhận thức. Cấu hình ban đầu của tác nhân có thể phản ánh 1 số kiến thức trước đó về môi trường, nhưng khi tác nhân có thêm kinh nghiệm, điều này có thể được sửa đổi & tăng cường. Có những trường hợp cực đoan trong đó môi trường được biết đến hoàn toàn *a priori* & hoàn toàn có thể dự đoán được. Trong những trường hợp như vậy, tác nhân không cần phải nhận thức hoặc học; nó chỉ đơn giản là hành động đúng.

Of course, such agents are fragile. Consider lowly dung beetle. After digging its nest & laying its eggs, it fetches a ball of dung from a nearby heap to plug entrance. If ball of dung is removed from its grasp *en route*, beetle continues its task & pantomimes plugging nest with nonexistent dung ball, never noticing that it is missing. Evolution has built an assumption into beetle's behavior, & when it is violated, unsuccessful behavior results.

– Tất nhiên, những tác nhân như vậy rất mong manh. Hãy xem xét loài bọ hung thấp kém. Sau khi đào tổ & đẻ trứng, nó lấy 1 cục phân từ đống phân gần đó để chặn lối vào. Nếu cục phân bị lấy khỏi tay nó trên đường đi, bọ hung tiếp tục nhiệm vụ & làm trò hề bằng cách chặn tổ bằng cục phân không tồn tại, không bao giờ nhận ra rằng cục phân đã mất. Sự tiến hóa đã xây dựng 1 giả định vào hành vi của bọ hung, & khi giả định đó bị vi phạm, hành vi không thành công sẽ xảy ra.

Slightly more intelligent is sphex wasp. Female sphex will dig a burrow, go out & sting a caterpillar & drag it to burrow, enter burrow again to check all is well, drag caterpillar inside, & lay its eggs. Caterpillar serves as a food source when eggs hatch. So far so good, but if an entomologist moves caterpillar a few inches away while sphex is doing check, it will revert to "drag caterpillar" step of its plan & will continue plan without modification, re–checking burrow, even after dozens of caterpillar-moving interventions. Sphex is unable to learn that its innate plan is failing, & thus will not change it.

– Thông minh hơn 1 chút là ong bắp cày sphex. Ong sphex cái sẽ đào hang, chui ra & đốt sâu bướm & kéo nó vào hang, chui vào hang lần nữa để kiểm tra mọi thứ ổn thỏa, kéo sâu bướm vào trong, & đẻ trứng. Sâu bướm đóng vai trò là nguồn thức ăn khi trứng nở. Cho đến giờ thì mọi thứ vẫn ổn, nhưng nếu 1 nhà côn trùng học di chuyển sâu bướm ra xa vài inch trong khi sphex đang kiểm tra, nó sẽ quay lại bước "kéo sâu bướm" trong kế hoạch của nó & sẽ tiếp tục kế hoạch mà không cần sửa đổi, kiểm tra lại hang, ngay cả sau hàng chục lần can thiệp di chuyển sâu bướm. Sphex không thể học được rằng kế hoạch bẩm sinh của nó đang thất bại, & do đó sẽ không thay đổi nó.

To extent that an agent relies on prior knowledge of its designer rather than on its own percepts & learning processes, say: agent lacks *autonomy*. A rational agent should be autonomous – it should learn what it can to compensate for partial or incorrect prior knowledge. E.g., a vacuum-cleaning agent that learns to predict where & when additional dirt will appear will do better than one that does not.

– Trong phạm vi mà 1 tác nhân dựa vào kiến thức trước đó của người thiết kế nó hơn là vào các nhận thức & quá trình học tập của chính nó, hãy nói: tác nhân thiếu *tự chủ*. 1 tác nhân hợp lý phải tự chủ – nó phải học những gì nó có thể để bù đắp cho kiến thức trước đó không đầy đủ hoặc không chính xác. Ví dụ, 1 tác nhân hút bụi học cách dự đoán nơi & khi nào bụi bẩn sẽ xuất hiện sẽ hoạt động tốt hơn 1 tác nhân không làm như vậy.

As a practical matter, one seldom requires complete autonomy from start: when agent has had little or no experience, it would have to act randomly unless designer gave some assistance. Just as evolution provides animals with enough built-in reflexes to survive long enough to learn for themselves, it would be reasonable to provide an AI agent with some initial knowledge as well as an ability to learn. After sufficient experience of its environment, behavior of a rational agent can become effectively *independent* of its prior knowledge. Hence, incorporation of learning allows one to design a single rational agent that will succeed in a vast variety of environments.

– Trên thực tế, người ta hiếm khi đòi hỏi sự tự chủ hoàn toàn ngay từ đầu: khi tác nhân có ít hoặc không có kinh nghiệm, nó sẽ phải hành động ngẫu nhiên trừ khi nhà thiết kế cung cấp 1 số hỗ trợ. Cũng giống như quá trình tiến hóa cung cấp cho động vật đủ phản xạ tích hợp để tồn tại đủ lâu để tự học, sẽ hợp lý khi cung cấp cho tác nhân AI 1 số kiến thức ban đầu cũng như khả năng học hỏi. Sau khi có đủ kinh nghiệm về môi trường của mình, hành vi của tác nhân hợp lý có thể trở nên *độc lập* hiệu quả với kiến thức trước đó của nó. Do đó, việc kết hợp học tập cho phép người ta thiết kế 1 tác nhân hợp lý duy nhất sẽ thành công trong nhiều môi trường khác nhau.

∗ 2.3. Nature of Environments. Now have a definition of rationality, almost ready to think about building rational agents. 1st, however, must think about *task environments*, which are essentially "problems" to which rational agents are "solutions". Begin by showing how to specify a task environment, illustrating process with a number of examples. Then show: task environments come in a variety of flavors. Nature of task environment directly affects appropriate design for agent program.

– *Bản chất của Môi trường.* Bây giờ đã có định nghĩa về tính hợp lý, gần như đã sẵn sàng để nghĩ về việc xây dựng các tác nhân hợp lý. Tuy nhiên, trước tiên, phải nghĩ về *task environments*, về cơ bản là "các vấn đề" mà các tác nhân hợp lý là "giải pháp". Bắt đầu bằng cách chỉ ra cách chỉ định 1 môi trường tác vụ, minh họa quy trình bằng 1 số ví dụ. Sau đó chỉ ra: các môi trường tác vụ có nhiều loại khác nhau. Bản chất của môi trường tác vụ ảnh hưởng trực tiếp đến thiết kế phù hợp cho chương trình tác nhân.

· 2.3.1. Specifying task environment. In our discussion of rationality of simple vacuum-cleaner agent, had to specify performance measure, environment, & agent's actuators & sensors. Group all these under heading of *task environment*. For acronymically minded, call this PEAS (Performance, Environment, Actuators, Sensors) description. In designing an agent, 1st step mus always be to specify task environment as fully as possible.

Vacuum world was a simple example; consider a more complex problem: an automated taxi driver. Fig. 2.4: PEAS description of task environment for an automated taxi driver. summarizes PEAS description for taxi's task environment. Discuss each element in more detail.

1st, what is *performance measure* to which we would like our automated driver to aspire? Desirable qualities include

getting to correct destination; minimizing fuel consumption & wear & tear; minimizing trip time or cost; minimizing violations of traffic laws & disturbances to other drivers; maximizing safety & passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.
– 1, *tiêu chuẩn hiệu suất* mà chúng ta muốn trình điều khiển tự động của mình hướng tới là gì? Các phẩm chất mong muốn bao gồm đến đúng đích; giảm thiểu mức tiêu thụ nhiên liệu & hao mòn & rách; giảm thiểu thời gian hoặc chi phí chuyến đi; giảm thiểu vi phạm luật giao thông & gây phiền nhiễu cho những người lái xe khác; tối đa hóa sự an toàn & sự thoải mái của hành khách; tối đa hóa lợi nhuận. Rõ ràng, 1 số mục tiêu này xung đột với nhau, vì vậy cần phải đánh đổi.
Next, what is driving *environment* that taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes & urban alleys to 12-lane freeways. Roads contain other traffic, pedestrians, stray animals, road works, police cars, puddles, & potholes. Taxi must also interact with potential & actual passengers. There are also some optional choices. Taxi might need to operate in Southern California, where snow is seldom a problem, or in Alaska, where it seldom is not. It could always driving on right, or we might want it to be flexible enough to drive on left when in Britain or Japan. Obviously, more restricted environment, easier design problem.
– Tiếp theo, môi trường lái xe mà taxi sẽ phải đối mặt là gì? Bất kỳ tài xế taxi nào cũng phải đối mặt với nhiều loại đường khác nhau, từ làn đường nông thôn & ngõ phố đến đường cao tốc 12 làn. Đường có nhiều phương tiện giao thông khác, người đi bộ, động vật hoang dã, công trình đường bộ, xe cảnh sát, vũng nước, & ổ gà. Taxi cũng phải tương tác với hành khách tiềm năng & thực tế. Ngoài ra còn có 1 số lựa chọn tùy chọn. Taxi có thể cần hoạt động ở Nam California, nơi tuyết hiếm khi là vấn đề, hoặc ở Alaska, nơi tuyết hiếm khi không là vấn đề. Nó luôn có thể lái xe bên phải, hoặc chúng ta có thể muốn nó đủ linh hoạt để lái xe bên trái khi ở Anh hoặc Nhật Bản. Rõ ràng, môi trường hạn chế hơn, vấn đề thiết kế dễ dàng hơn.
Actuators for an automated taxi include those available to a human driver: control over engine through accelerator & control over steering & braking. In addition, it will need output to a display screen or voice synthesizer to talk back to passengers, & perhaps some way to communicate with other vehicles, politely or otherwise.
– *Bộ truyền động* cho 1 chiếc taxi tự động bao gồm những bộ truyền động có sẵn cho người lái: điều khiển động cơ thông qua chân ga & điều khiển tay lái & phanh. Ngoài ra, nó sẽ cần xuất ra màn hình hiển thị hoặc bộ tổng hợp giọng nói để nói chuyện với hành khách, & có lẽ là 1 số cách để giao tiếp với các phương tiện khác, theo cách lịch sự hoặc không.
Basic *sensors* for taxi will include 1 or more video cameras so that it can see, as well as lidar & ultrasound sensors to detect distances to other cars & obstacles. To avoid speeding tickets, taxi should have a speedometer, & to control vehicle properly, especially on curves, it should have an accelerometer. To determine mechanical state of vehicle, it will need usual array of engine, fuel, & electrical system sensors. Like many human drivers, it might want to access GPT signals so that it doesn't get lost. Finally, it will need touchscreen or voice input for passenger to request a destination.
– Các cảm biến cơ bản của taxi sẽ bao gồm 1 hoặc nhiều camera video để có thể nhìn thấy, cũng như cảm biến lidar & siêu âm để phát hiện khoảng cách đến các xe khác & chướng ngại vật. Để tránh bị phạt vì chạy quá tốc độ, taxi nên có đồng hồ đo tốc độ, & để điều khiển xe đúng cách, đặc biệt là khi vào cua, taxi nên có máy đo gia tốc. Để xác định trạng thái cơ học của xe, taxi sẽ cần 1 loạt các cảm biến thông thường về động cơ, nhiên liệu, & hệ thống điện. Giống như nhiều tài xế khác, taxi có thể muốn truy cập tín hiệu GPT để không bị lạc đường. Cuối cùng, taxi sẽ cần màn hình cảm ứng hoặc giọng nói để hành khách yêu cầu điểm đến.
In Fig. 2.5: Examples of agent types & their PEAS descriptions., have sketched basic PEAS elements for a number of additional agent types. Further examples appear in Exercise 2.PEAS. Examples include physical as well as virtual environments. Note: virtual task environments can be just as complex as "real" world: e.g., a *software agent* (or software robot or *softbot*) that trades on auction & reselling Web sites deals with millions of other users & billions of objects, many with real images.
– Trong Hình 2.5: Ví dụ về các loại tác nhân & mô tả PEAS của chúng., đã phác thảo các thành phần PEAS cơ bản cho 1 số loại tác nhân bổ sung. Các ví dụ khác xuất hiện trong Bài tập 2.PEAS. Các ví dụ bao gồm cả môi trường vật lý cũng như môi trường ảo. Lưu ý: môi trường tác vụ ảo có thể phức tạp như thế giới "thực": ví dụ, 1 *phần mềm tác nhân* (hoặc rô-bốt phần mềm hoặc *softbot*) giao dịch trên các trang web đấu giá & bán lại giao dịch với hàng triệu người dùng khác & hàng tỷ đối tượng, nhiều đối tượng có hình ảnh thực.

· 2.3.2. Properties of task environments. Range of task environments that might arise in AI is obviously vast. Can, however, identify a fairly small number of dimensions along which task environments can be categorized. These dimensions determine, to a large extent, appropriate agent design & applicability of each of principal families of techniques for agent implementation. 1st list dimensions, then analyze several task environments to illustrate ideas. Definitions here are informal; later chaps provide more precise statements & examples of each kind of environment.
– *Thuộc tính của môi trường tác vụ.* Phạm vi các môi trường tác vụ có thể phát sinh trong AI rõ ràng là rất lớn. Tuy nhiên, có thể xác định 1 số lượng khá nhỏ các chiều mà môi trường tác vụ có thể được phân loại theo. Các chiều này xác định, ở 1 mức độ lớn, thiết kế tác nhân phù hợp & khả năng áp dụng của từng họ kỹ thuật chính để triển khai tác nhân. Đầu tiên, hãy liệt kê các chiều, sau đó phân tích 1 số môi trường tác vụ để minh họa các ý tưởng. Các định nghĩa ở đây là không chính thức; các chương sau cung cấp các tuyên bố chính xác hơn & ví dụ về từng loại môi trường.
p. 61+++

◦

PART II: PROBLEM-SOLVING.

◦ 3. Solving Problems by Searching.

- 4. Search in Complex Environments.
- 5. Constraint Satisfaction Problems.
- 6. Adversarial Search & Games.

PART III: KNOWLEDGE, REASONING, & PLANNING.

- 7. Logical Agents.
- 8. 1st-Order Logic.
- 9. Inference in 1st-Order Logic.
- 10. Knowledge Representation.
- 11. Automated Planning.

PART IV: UNCERTAIN KNOWLEDGE & REASONING.

- 12. Quantifying Uncertainty.
- 13. Probabilistic Reasoning.
- 14. Probabilistic Reasoning over Time.
- 15. Making Simple Decisions.
- 16. Making Complex Decisions.
- 17. Multiagent Decision Making.
- 18. Probabilistic Programming.

PART V: ML

- 19. Learning from Examples.
- 20. Knowledge in Learning.
- 21. Learning Probabilistic Models.
- 22. DL.
- 23. Reinforcement Learning.

PART VI: COMMUNICATING, PERCEIVING, & ACTING.

- 24. Natural Language Processing.
- 25. DL for NLP.
- 26. Robotics.
- 27. Computer Vision.

PART VII: CONCLUSIONS.

- 28. Philosophy, Ethics, & Safety of AI.
- 29. Future of AI.

- Appendix A: Mathematical Background.
- Appendix B: Notes on Languages & Algorithms.

2 KnapSack Problem

2.1 [KPP04]. Hans Kellerer, Ulrich Pferschy, David Pisinger. Knapsack Problems

- Preface. 30 years have passed since seminal book on knapsack problems by Martello & Toth appeared. On this occasion a former colleague exclaimed back in 1990: "How can you write 250 pages on knapsack problem?" Indeed, def of knapsack problem is easily understood even by a non-expert who will not suspect presence of challenging research topics in this area at 1st glance.

  However, in last decade a large number of research publications contributed new results for knapsack problem in all areas of interest e.g. exact algorithms, heuristics, & approximation schemes. Moreover, extension of knapsack problem to higher dimensions both in number of constraints & in number of knapsacks, as well as modification of problem structure concerning available item seet & objective function, leads to a number of interesting variations of practical relevance which were subject of intensive research during last few years.

  – Tuy nhiên, trong thập kỷ qua, 1 số lượng lớn các ấn phẩm nghiên cứu đã đóng góp những kết quả mới cho bài toán ba lô trong mọi lĩnh vực quan tâm, ví dụ như thuật toán chính xác, phương pháp tìm kiếm, & sơ đồ xấp xỉ. Hơn nữa, việc mở rộng

bài toán ba lô lên các chiều cao hơn về cả số lượng ràng buộc & số lượng ba lô, cũng như việc sửa đổi cấu trúc bài toán liên quan đến mục có sẵn & hàm mục tiêu, dẫn đến 1 số biến thể thú vị có liên quan thực tế, là chủ đề của nghiên cứu chuyên sâu trong vài năm qua.

Hence, 2 years ago idea arose to produce a new monograph covering not only most recent developments of standard knapsack problem, but also giving a comprehensive treatment of whole knapsack family including siblings e.g. subset sum problem & bounded & unbounded knapsack problem, & also more distant relatives e.g. multidimensional, multiple, multiple-choice & quadratic knapsack problems in dedicated chaps.

– Do đó, 2 năm trước, 1 ý tưởng đã nảy sinh là biên soạn 1 chuyên khảo mới không chỉ bao gồm những phát triển gần đây nhất của bài toán ba lô chuẩn mà còn đưa ra cách xử lý toàn diện cho toàn bộ họ ba lô bao gồm cả các phần tử cùng nhóm, ví dụ như bài toán tổng tập hợp & bài toán ba lô có giới hạn & không giới hạn, & cả những họ hàng xa hơn, ví dụ như bài toán ba lô đa chiều, nhiều, nhiều lựa chọn & bậc hai trong các chương chuyên đề.

Furthermore, attention is paid to a number of less frequently considered variants of knapsack problem & to study of stochastic aspects of problem. To illustrate high practical relevance of knapsack family for many industrial & economic problems, a number of applications are described in more detail. They are selected subjectively from innumerable occurrences of knapsack problems reported in literature.

– Hơn nữa, sự chú ý được dành cho 1 số biến thể ít được xem xét hơn của bài toán ba lô & để nghiên cứu các khía cạnh ngẫu nhiên của bài toán. Để minh họa tính liên quan thực tế cao của họ ba lô đối với nhiều bài toán công nghiệp & kinh tế, 1 số ứng dụng được mô tả chi tiết hơn. Chúng được lựa chọn 1 cách chủ quan từ vô số trường hợp xảy ra của bài toán ba lô được báo cáo trong tài liệu.

Our above-mentioned colleague will be surprised to notice that even on > 500 pages of this book not all relevant topics could be treated in equal depth but decisions had to be made on where to go into details of constructions & proofs & where to concentrate on stating results & refer to appropriate publications. Moreover, an editorial deadline had to be drawn at some point. In our case, stopped looking for new publications at end of Jun 2003.

– Người đồng nghiệp được đề cập ở trên của chúng tôi sẽ ngạc nhiên khi nhận thấy rằng ngay cả trên > 500 trang của cuốn sách này, không phải tất cả các chủ đề có liên quan đều có thể được xử lý ở mức độ sâu như nhau mà phải đưa ra quyết định về việc đi sâu vào chi tiết về các cấu trúc & bằng chứng & tập trung vào việc nêu kết quả & tham khảo các ấn phẩm phù hợp. Hơn nữa, phải đưa ra thời hạn biên tập tại 1 thời điểm nào đó. Trong trường hợp của chúng tôi, đã ngừng tìm kiếm các ấn phẩm mới vào cuối tháng 6 năm 2003.

Audience we envision for this book is 3fold: 1st 2 chaps offer a very basic introduction to knapsack problem & main algorithmic concepts to derive optimal & approximate solution. Chap. 3 presents a number of advanced algorithmic techniques which are used throughout later chaps of book. Style of presentation in these 3 chaps is kept rather simple & assumes only minimal prerequisites. They should be accessible to students & graduates of business administration, economics & engineering as well as practitioners with little knowledge of algorithms & optimization.

– Đối tượng mà chúng tôi hình dung cho cuốn sách này là 3 phần: 2 chương đầu tiên cung cấp phần giới thiệu rất cơ bản về bài toán ba lô & các khái niệm thuật toán chính để đưa ra giải pháp tối ưu & gần đúng. Chương 3 trình bày 1 số kỹ thuật thuật toán nâng cao được sử dụng trong các chương sau của cuốn sách. Phong cách trình bày trong 3 chương này được giữ khá đơn giản & chỉ giả định các điều kiện tiên quyết tối thiểu. Chúng nên dễ tiếp cận đối với sinh viên & tốt nghiệp ngành quản trị kinh doanh, kinh tế & kỹ thuật cũng như những người hành nghề có ít kiến thức về thuật toán & tối ưu hóa.

This 1st part of book is also well suited to introduce classical concepts of optimization in a classroom, since knapsack problem is easy to understand & is probably the least difficult but most illustrative problem where dynamic programming, branch-&-bound, relaxations & approximation schemes can be applied.

– Phần 1 của cuốn sách này cũng rất phù hợp để giới thiệu các khái niệm cổ điển về tối ưu hóa trong lớp học, vì bài toán ba lô dễ hiểu & có lẽ là bài toán ít khó nhất nhưng minh họa rõ nhất, trong đó có thể áp dụng các lược đồ lập trình động, nhánh&-bound, nới lỏng & xấp xỉ.

In these chaps no knowledge of linear or integer programming & only a minimal familiarity with basic elements of graph theory is assumed. Issue of NP-completeness is dealt with by an intuitive introduction in Sect. 1.5, whereas a thorough & rigorous treatment is deferred to Appendix.

– Trong các chương này không có kiến thức về lập trình tuyến tính hoặc số nguyên & chỉ giả định có sự quen thuộc tối thiểu với các yếu tố cơ bản của lý thuyết đồ thị. Vấn đề về tính đầy đủ của NP được giải quyết bằng phần giới thiệu trực quan trong Phần 1.5, trong khi phần xử lý kỹ lưỡng & chặt chẽ được chuyển đến Phụ lục.

Remaining chaps of book addresses 2 different audiences. On 1 hand, a student or graduate of mathematics or CS, or a successful reader of 1st 3 chaps willing to go into more depth, can use this book to study advanced algorithms for knapsack problem & its relatives. On other hand, hope scientific researchers or expert practitioners will find book a valuable source of reference for a quick update on state of art & on most efficient algorithms currently available. In particular, a collection of computational experiments, many of them published for 1st time in this book, should serve as a valuable tool to pick algorithm best suited for a given problem instance. To facilitate use of book as a reference, tried to keep these chaps self-contained as far as possible.

– Các chương còn lại của cuốn sách hướng đến 2 đối tượng độc giả khác nhau. 1 mặt, 1 sinh viên hoặc tốt nghiệp ngành toán học hoặc khoa học máy tính, hoặc 1 độc giả thành công của 3 chương đầu tiên muốn đi sâu hơn, có thể sử dụng cuốn sách

này để nghiên cứu các thuật toán nâng cao cho bài toán ba lô & các chương liên quan. Mặt khác, hy vọng các nhà nghiên cứu khoa học hoặc các chuyên gia thực hành sẽ thấy cuốn sách là 1 nguồn tham khảo có giá trị để cập nhật nhanh về tình hình nghệ thuật & các thuật toán hiệu quả nhất hiện có. Đặc biệt, 1 bộ sưu tập các thí nghiệm tính toán, nhiều trong số chúng được công bố lần đầu tiên trong cuốn sách này, sẽ đóng vai trò là 1 công cụ có giá trị để chọn thuật toán phù hợp nhất cho 1 trường hợp bài toán nhất định. Để tạo điều kiện thuận lợi cho việc sử dụng cuốn sách làm tài liệu tham khảo, chúng tôi đã cố gắng giữ các chương này độc lập nhất có thể.

For these advanced audiences assume familiarity with basic theory of linear programming, elementary elements of graph theory, & concepts of algorithms & data structures as far as they are generally taught in basic courses on these subjects.

– Đối với những đối tượng nâng cao này, giả sử họ đã quen thuộc với lý thuyết cơ bản về lập trình tuyến tính, các yếu tố cơ bản của lý thuyết đồ thị, & các khái niệm về thuật toán & cấu trúc dữ liệu theo như những gì thường được dạy trong các khóa học cơ bản về các chủ đề này.

Chaps. 4–12 give detailed presentations of knapsack problem & its variants in increasing order of structural difficulty. Hence, start with subset sum problem in Chap. 4, move on to standard knapsack problem which is discussed extensively in 2 chaps, 1 for exact & 1 for approximate algorithms, & finish this 2nd part of book with bounded & unbounded knapsack problem in Chaps. 7–8.

3rd part of book contains more complicated generalizations of knapsack problems. It starts with multidimensional knapsack problem (a knapsack problem with d constraints) in Chap. 9, then considers multiple knapsack problem (m knapsacks are available for packing) in Chap. 10, goes on to multiple-choice knapsack problem (items are partitioned into classes & exactly 1 item of each class must be packed), & extends linear objective function to a quadratic one yielding quadratic knapsack problem in Chap. 12. This chap also contains an excursion to semidefinite programming giving a mostly self-contained short introduction to this topic.

– Phần thứ 3 của cuốn sách bao gồm những khái quát phức tạp hơn về các bài toán ba lô. Nó bắt đầu với bài toán ba lô đa chiều (một bài toán ba lô với d ràng buộc) trong Chương 9, sau đó xem xét nhiều bài toán ba lô (m ba lô có sẵn để đóng gói) trong Chương 10, tiếp tục đến bài toán ba lô trắc nghiệm (các mục được phân vùng thành các lớp & chính xác 1 mục của mỗi lớp phải được đóng gói), & mở rộng hàm mục tiêu tuyến tính thành 1 hàm bậc hai tạo ra bài toán ba lô bậc hai trong Chương 12. Chương này cũng bao gồm 1 chuyến tham quan đến lập trình bán xác định cung cấp phần giới thiệu ngắn gọn chủ yếu là độc lập về chủ đề này.

All these 6 chaps can be seen as survey articles, most of them being 1st survey on their subject, containing many pointers to literature & some examples of application.

Particular effort was put into description of interesting applications of knapsack type problems. Decided to avoid a boring listing of umpteen papers with a 2-line description of occurrence of a knapsack problem for each of them, but selected a smaller number of application areas where knapsack models play a prominent role. These areas are discussed in more detail in Chap. 15 to give reader a full understanding of situations presented. They should be particularly useful for teaching purposes.

– Nỗ lực đặc biệt đã được đưa vào việc mô tả các ứng dụng thú vị của các bài toán loại ba lô. Quyết định tránh 1 danh sách nhàm chán của hàng tá bài báo với mô tả 2 dòng về sự xuất hiện của 1 bài toán ba lô cho mỗi bài báo, nhưng đã chọn 1 số ít các lĩnh vực ứng dụng mà các mô hình ba lô đóng vai trò nổi bật. Các lĩnh vực này được thảo luận chi tiết hơn trong Chương 15 để cung cấp cho người đọc sự hiểu biết đầy đủ về các tình huống được trình bày. Chúng sẽ đặc biệt hữu ích cho mục đích giảng dạy.

Appendix gives a short presentation of NP-completeness with focus on knapsack problems. Without venturing into depths of theoretical CS & avoiding topics e.g. Turing machines & unary encoding, a rather informal introduction to NP-completeness is given, however with formal proofs for NP-hardness of subset sum & knapsack problem.

– Phụ lục trình bày ngắn gọn về NP-completeness tập trung vào các bài toán knapsack. Không đi sâu vào CS lý thuyết & tránh các chủ đề như máy Turing & mã hóa đơn phân, phần giới thiệu khá không chính thức về NP-completeness được đưa ra, tuy nhiên có các bằng chứng chính thức cho NP-hardness của tổng tập con & bài toán knapsack.

Some assumptions & conventions concerning notation & style are kept throughout book. Most algorithms are stated in a flexible pseudocode style putting emphasis on readability instead of formal uniformity. I.e., simpler algorithms are given in style of an known but easily understandable programming language, whereas more complex algorithms are introduced by a structured, but verbal description. Commands & verbal instructions are given in Sans Serif font, whereas comments follow in *Italic* letters. As a general reference & guideline to algorithms we used book by Cormen, Leiserson, Rivest & Stein [92].

– 1 số giả định & quy ước liên quan đến ký hiệu & phong cách được giữ nguyên trong toàn bộ cuốn sách. Hầu hết các thuật toán được nêu theo phong cách mã giả linh hoạt, nhấn mạnh vào khả năng đọc được thay vì tính thống nhất về mặt hình thức. Tức là, các thuật toán đơn giản hơn được đưa ra theo phong cách của 1 ngôn ngữ lập trình đã biết nhưng dễ hiểu, trong khi các thuật toán phức tạp hơn được giới thiệu bằng mô tả có cấu trúc nhưng bằng lời. Các lệnh & hướng dẫn bằng lời được đưa ra bằng phông chữ Sans Serif, trong khi các bình luận theo sau bằng chữ *Italic*. Là tài liệu tham khảo chung & hướng dẫn cho các thuật toán, chúng tôi đã sử dụng cuốn sách của Cormen, Leiserson, Rivest & Stein [92].

For sake of readability & personal taste follow non-standard convention of using term *increasing* instead of mathematically correct *nondecreasing* & in same way *decreasing* instead of *nonincreasing*. Whereever use log function always refer to base 2 logarithm unless stated otherwise. After Preface give a short list of notations containing only those terms which are used throughout book. Many more naming conventions will be introduced on a local level during individual chaps & sects.

A number of computational experiments were performed for exact algorithms. These were performed on following machines: AMD Athlon, Intel Pentium 4, III. Performance index was obtained from SPEC www.specbench.org. As can be seen 3 machines have reasonably similar performance, making it possible to compare running times across chaps. Codes have been compiled using GNU project C & C++ Compiler gcc-2.96, which also compiles Fortran77 code, thus preventing differences in computation times due to alternative compilers.

– 1 số thí nghiệm tính toán đã được thực hiện cho các thuật toán chính xác. Những thí nghiệm này được thực hiện trên các máy sau: AMD Athlon, Intel Pentium 4, III. Chỉ số hiệu suất được lấy từ SPEC www.specbench.org. Như có thể thấy, 3 máy có hiệu suất khá giống nhau, giúp có thể so sánh thời gian chạy giữa các chap. Các mã đã được biên dịch bằng dự án GNU C & C++ Compiler gcc-2.96, cũng biên dịch mã Fortran77, do đó ngăn ngừa sự khác biệt về thời gian tính toán do các trình biên dịch thay thế.

- Acknowledgments. Authors strongly beieve in necessity to do research not with an island mentality but in an open exchange of knowledge, opinions & ideas within an international research community. Clearly, none of us would have been able to contribute to this book without innumerable personal exchanges with colleagues on conferences & workshops, in person, by email or even by surface mail. Therefore, like to start our acknowledgments by thanking global research community for providing spirit necessary for joint projects of collection & presentation. Importance of solving knapsack problems for all values of capacity.

  – Các tác giả tin tưởng mạnh mẽ vào sự cần thiết phải nghiên cứu không phải với tâm lý đảo mà trong sự trao đổi cởi mở về kiến thức, ý kiến & ý tưởng trong cộng đồng nghiên cứu quốc tế. Rõ ràng, không ai trong chúng ta có thể đóng góp cho cuốn sách này nếu không có vô số cuộc trao đổi cá nhân với các đồng nghiệp tại các hội nghị & hội thảo, trực tiếp, qua email hoặc thậm chí qua thư. Do đó, chúng tôi muốn bắt đầu lời cảm ơn của mình bằng cách cảm ơn cộng đồng nghiên cứu toàn cầu đã cung cấp tinh thần cần thiết cho các dự án chung về thu thập & trình bày. Tầm quan trọng của việc giải quyết các vấn đề ba lô cho tất cả các giá trị của năng lực.

- 1. Introduction.

  - 1.1. Introducing Knapsack Problem. Every aspect of human life is crucially determined by result of decisions. Whereas private decisions may be based on emotions or personal taste, complex professional environment of 21st century requires a decision process which can be formalized & validated independently from involved individuals. Therefore, a quantitative formulation of all factors influencing a decision & also of result of decision process is sought.

    – Mọi khía cạnh của cuộc sống con người đều được quyết định 1 cách quan trọng bởi kết quả của các quyết định. Trong khi các quyết định riêng tư có thể dựa trên cảm xúc hoặc sở thích cá nhân, môi trường chuyên nghiệp phức tạp của thế kỷ 21 đòi hỏi 1 quy trình ra quyết định có thể được chính thức hóa & xác thực độc lập với các cá nhân liên quan. Do đó, cần tìm kiếm 1 công thức định lượng của tất cả các yếu tố ảnh hưởng đến quyết định & cũng như kết quả của quy trình ra quyết định.

    In order to meet this goal it must be possible to represent effect of any decision by numerical values. In most basic case outcome of decision can be measured by a single value representing gain, profit, loss, cost or some other category of data. Comparison of these values induces a total order on set of all options which are available for a decision. Finding option with highest or lowest value can be difficult because set of available options may be extremely large &/or not explicitly known. Frequently, only conditions are known which characterize feasible options out of a very general ground set of theoretically available choices.

    – Để đạt được mục tiêu này, phải có thể biểu diễn hiệu ứng của bất kỳ quyết định nào bằng các giá trị số. Trong hầu hết các trường hợp cơ bản, kết quả của quyết định có thể được đo bằng 1 giá trị duy nhất biểu diễn mức tăng, lợi nhuận, tổn thất, chi phí hoặc 1 số loại dữ liệu khác. So sánh các giá trị này tạo ra thứ tự tổng thể trên tập hợp tất cả các tùy chọn có sẵn cho 1 quyết định. Việc tìm ra tùy chọn có giá trị cao nhất hoặc thấp nhất có thể khó khăn vì tập hợp các tùy chọn có sẵn có thể cực kỳ lớn &hoặc không được biết rõ ràng. Thông thường, chỉ có các điều kiện được biết là đặc trưng cho các tùy chọn khả thi trong 1 tập hợp cơ sở rất chung của các lựa chọn có sẵn về mặt lý thuyết.

    Simplest possible form of a decision is choice between 2 alternatives. Such a *binary decision* is formulated in a quantitative model as a *binary variable* $x \in \{0, 1\}$ with obvious meaning that $x = 1$ means taking 1st alternative whereas $x = 0$ indicates rejection of 1st alternative & hence selection of 2nd option.

    – Dạng đơn giản nhất có thể của 1 quyết định là lựa chọn giữa 2 phương án thay thế. 1 *quyết định nhị phân* như vậy được xây dựng trong 1 mô hình định lượng như 1 *biến nhị phân* $x \in \{0, 1\}$ với ý nghĩa rõ ràng là $x = 1$ có nghĩa là chọn phương án thay thế thứ nhất trong khi $x = 0$ biểu thị việc từ chối phương án thay thế thứ nhất & do đó lựa chọn phương án thứ 2.

    Many practical decision processes can be represented by an appropriate combination of several binary decisions. I.e., overall decision problem consists of choosing 1 of 2 alternatives for a large number of binary decisions which may all influence each other. In basic version of a *linear decision model* outcome of complete decision process is evaluated by a linear combination of values associated with each of binary decisions. In order to take pairwise interdependencies between decisions into account also a *quadratic function* can be used to represent outcome of a decision process. Feasibility of a particular selection of alternatives may be very complicated to establish in practice because binary decisions may influence or even contract each other.

    – Nhiều quá trình ra quyết định thực tế có thể được biểu diễn bằng sự kết hợp thích hợp của 1 số quyết định nhị phân. Tức là, vấn đề quyết định tổng thể bao gồm việc lựa chọn 1 trong 2 phương án thay thế cho 1 số lượng lớn các quyết định nhị phân mà tất cả đều có thể ảnh hưởng lẫn nhau. Trong phiên bản cơ bản của *mô hình quyết định tuyến tính*, kết quả của toàn bộ quá trình ra quyết định được đánh giá bằng sự kết hợp tuyến tính của các giá trị liên quan đến từng quyết định

nhị phân. Để tính đến sự phụ thuộc lẫn nhau theo cặp giữa các quyết định, cũng có thể sử dụng *hàm bậc hai* để biểu diễn kết quả của 1 quá trình ra quyết định. Tính khả thi của 1 lựa chọn thay thế cụ thể có thể rất phức tạp để thiết lập trong thực tế vì các quyết định nhị phân có thể ảnh hưởng hoặc thậm chí thu hẹp lẫn nhau.

Formally speaking, linear decision model is defined by n binary variables $x_j \in \{0,1\}$ which correspond to selection in jth binary decision & by *profit values* p_j which indicate difference of value attained by choosing 1st alternative, i.e., $x_j = 1$, instead of 2nd alternative $x_j = 0$. W.l.o.g. can assume: after a suitable assignment of 2 options to 2 cases $x_j = 1$ & $x_j = 0$, always have $p_j \geq 0$. Overall profit value associated with a particular choice $\forall n$ binary decisions is given by sum of all values p_j for all decisions where 1st alternative was selected.

– Về mặt hình thức, mô hình quyết định tuyến tính được định nghĩa bởi n biến nhị phân $x_j \in \{0,1\}$ tương ứng với lựa chọn trong quyết định nhị phân jth & bởi *giá trị lợi nhuận* p_j biểu thị sự khác biệt về giá trị đạt được khi chọn phương án thứ nhất, tức là $x_j = 1$, thay vì phương án thứ hai $x_j = 0$. W.l.o.g. có thể giả sử: sau khi gán phù hợp 2 tùy chọn cho 2 trường hợp $x_j = 1$ & $x_j = 0$, luôn có $p_j \geq 0$. Tổng giá trị lợi nhuận liên quan đến 1 lựa chọn cụ thể $\forall n$ quyết định nhị phân được đưa ra bởi tổng của tất cả các giá trị p_j cho tất cả các quyết định trong đó phương án thứ nhất được chọn.

Consider decision problems where feasibility of a particular selection of alternatives can be evaluated by a linear combination of coefficients for each binary decision. In this model feasibility of a selection of alternatives is determined by a *capacity restriction* in following way. In every binary decision j selection of 1st alternative $x_j = 1$ requires a *weight* or *resource* w_j whereas choosing 2nd alternative $x_j = 0$ does not. A selection of alternatives is feasible if sum of weights over all binary decisions does not exceed a given threshold capacity value c. This condition can be written as $\sum_{i=1}^{n} w_i x_i \leq c$. Considering this decision process as an optimization problem, where overall profit should be as large as possible, yields *knapsack problem* (KP), core problem of this book.

– Hãy xem xét các bài toán quyết định trong đó tính khả thi của 1 lựa chọn thay thế cụ thể có thể được đánh giá bằng tổ hợp tuyến tính các hệ số cho mỗi quyết định nhị phân. Trong mô hình này, tính khả thi của 1 lựa chọn thay thế được xác định bằng *hạn chế năng lực* theo cách sau. Trong mọi quyết định nhị phân j, lựa chọn phương án thứ nhất $x_j = 1$ yêu cầu *trọng số* hoặc *tài nguyên* w_j trong khi việc chọn phương án thứ hai $x_j = 0$ thì không. 1 lựa chọn thay thế là khả thi nếu tổng trọng số trên tất cả các quyết định nhị phân không vượt quá giá trị ngưỡng năng lực c đã cho. Điều kiện này có thể được viết là $\sum_{i=1}^{n} w_i x_i \leq c$. Xem xét quá trình quyết định này như 1 bài toán tối ưu hóa, trong đó lợi nhuận tổng thể phải lớn nhất có thể, sẽ tạo ra *bài toán ba lô* (KP), bài toán cốt lõi của cuốn sách này.

This characteristic of problem gives rise to following interpretation of (KP) which is more colorful than combination of binary decision problems. Consider a mountaineer who is packing his knapsack (or rucksack) for a mountain tour & has to decide which items he should take with him. He has a large number of objects available which may be useful on his tour. Each of these items numbered from 1 to n would give him a certain amount of comfort or benefit which is measured by a positive number p_j. Of course, weight w_j of every object which mountaineer puts into his knapsack increases load he has to carry. For obvious reasons, he wants to limit total weight of his knapsack & hence fixes maximum load by capacity value c.

– Đặc điểm này của bài toán dẫn đến cách giải thích sau đây về (KP) có nhiều màu sắc hơn so với việc kết hợp các bài toán quyết định nhị phân. Hãy xem xét 1 người leo núi đang đóng gói ba lô (hoặc ba lô du lịch) của mình cho 1 chuyến leo núi & phải quyết định những vật dụng nào anh ta nên mang theo. Anh ta có 1 số lượng lớn các vật dụng có sẵn có thể hữu ích trong chuyến đi của mình. Mỗi vật dụng trong số này được đánh số từ 1 đến n sẽ mang lại cho anh ta 1 lượng thoải mái hoặc lợi ích nhất định được đo bằng 1 số dương p_j. Tất nhiên, trọng lượng w_j của mỗi vật dụng mà người leo núi để vào ba lô của mình sẽ làm tăng tải trọng mà anh ta phải mang. Vì những lý do hiển nhiên, anh ta muốn giới hạn tổng trọng lượng của ba lô của mình & do đó cố định tải trọng tối đa theo giá trị sức chứa c.

In order to give a more intuitive presentation, use this knapsack interpretation & usually refer to a "packing of items into a knapsack" rather than to "combination of binary decisions" throughout this book. Also terms "profit" & "weight" are based on this interpretation. Instead of making a number of binary decisions will speak of selection of a subset of items from *item set* $N := [n]$.

– Để đưa ra 1 bài trình bày trực quan hơn, hãy sử dụng cách diễn giải ba lô này & thường ám chỉ "đóng gói các mặt hàng vào ba lô" thay vì "kết hợp các quyết định nhị phân" trong suốt cuốn sách này. Ngoài ra, các thuật ngữ "lợi nhuận" & "trọng số" cũng dựa trên cách diễn giải này. Thay vì đưa ra 1 số quyết định nhị phân sẽ nói về việc lựa chọn 1 tập hợp con các mặt hàng từ *item set* $N := [n]$.

Knapsack problem (KP) can be formally defined as follows: We are given an *instance* of knapsack problem with item set N, consisting of n *items* j with *profit* p_j & *weight* w_j, & *capacity value* c. (Usually, all these values are taken from positive integer numbers.) Then objective: select a subset of N s.t. total profit of selected items is maximized & total weight does not exceed c.

Alternatively, a knapsack problem can be formulated as a solution of linear integer programming formulation:

$$(KP)\ \max \sum_{i=1}^{n} p_i x_i \text{ subject to } \sum_{i=1}^{n} w_i x_i \leq c,\ x_i \in \{0,1\}\, \forall j \in [n].$$

Denote *optimal solution vector* by $x^* = (x_1^*, \ldots, x_n^*)$ & *optimal solution value* by z^*. Set X^* denotes *optimal solution set*, i.e., set of items corresponding to optimal solution vector.

Problem (KP) is simplest nontrivial integer programming model with binary variables, only 1 single constraint & only positive coefficients. Nevertheless, adding integrality condition (1.3) $x_i \in \{0,1\}\, \forall j \in [n]$ to simple linear program (1.1)–(1.2) already puts (KP) into class of "difficult" problems.

– Bài toán (KP) là mô hình lập trình số nguyên phi tầm thường đơn giản nhất với các biến nhị phân, chỉ có 1 ràng buộc duy nhất & chỉ có hệ số dương. Tuy nhiên, việc thêm điều kiện tích phân (1.3) $x_i \in \{0,1\}\, \forall j \in [n]$ vào chương trình tuyến tính đơn giản (1.1)–(1.2) đã đưa (KP) vào lớp các bài toán "khó".

Knapsack problem has been studied for centuries as it is simplest prototype of a maximization problem. Already in 1897 Mathews [338] showed how several constraints may be aggregated into 1 single knapsack constraint. This is somehow a prototype of a reduction of a general integer program to (KP), thus proving: (KP is at least as hard to solve as an integer program. It is however unclear how name "Knapsack Problem" was invented. DANTZIG is using expression in his early work & thus name could be a kind of folklore.

– Bài toán ba lô đã được nghiên cứu trong nhiều thế kỷ vì đây là nguyên mẫu đơn giản nhất của bài toán tối đa hóa. Ngay từ năm 1897, Mathews [338] đã chỉ ra cách tổng hợp nhiều ràng buộc thành 1 ràng buộc ba lô duy nhất. Đây bằng cách nào đó là nguyên mẫu của phép rút gọn 1 chương trình số nguyên tổng quát thành (KP), do đó chứng minh rằng: (KP khó giải quyết ít nhất cũng như 1 chương trình số nguyên. Tuy nhiên, vẫn chưa rõ tên "Bài toán ba lô" được phát minh như thế nào. DANTZIG đang sử dụng cách diễn đạt trong tác phẩm đầu của mình & do đó tên có thể là 1 loại văn hóa dân gian.

Consider above characteristic of mountaineer in context of business instead of leisure leads to a 2nd classical interpretation of (KP) as an investment problem. A wealthy individual or institutional investor has a certain amount of money c available which she wants to put into profitable business projects. As a basis for her decisions she compiles a long list of possible investments including for every investment required amount w_i & expected net return p_i over a fixed period. Aspect of risk is not explicitly taken into account here. Obviously, combination of binary decisions for every investment s.t. overall return on investment is as large as possible can be formulated by (KP).

– Xem xét đặc điểm trên của người leo núi trong bối cảnh kinh doanh thay vì giải trí dẫn đến cách giải thích cổ điển thứ 2 của (KP) như 1 vấn đề đầu tư. 1 cá nhân giàu có hoặc nhà đầu tư tổ chức có 1 số tiền nhất định c có sẵn mà cô ấy muốn đầu tư vào các dự án kinh doanh có lợi nhuận. Để làm cơ sở cho các quyết định của mình, cô ấy biên soạn 1 danh sách dài các khoản đầu tư khả thi bao gồm cho mọi khoản đầu tư cần thiết là w_i & lợi nhuận ròng dự kiến p_i trong 1 khoảng thời gian cố định. Khía cạnh rủi ro không được tính đến 1 cách rõ ràng ở đây. Rõ ràng, sự kết hợp của các quyết định nhị phân cho mọi khoản đầu tư s.t. tổng lợi nhuận đầu tư lớn nhất có thể có thể được xây dựng bằng (KP).

A 3rd illustrating example of a real-world economic situation which is captured by (KP) is taken from airline cargo business. Dispatcher of a cargo airline has to decide which of transportation requests posed by customers he should fulfill, i.e., how to load a particular plane. His decision is based on a list of requests which contain weight w_i of every package & rate per weight unit charged for each request. Note: this rate is not fixed but depends on particular long-term arrangements with every customer. Hence profit p_i made by company by accepting a request & by putting corresponding package on plane is not directly proportional to weight of package. Naturally, every plane has a specified maximum capacity c which may not be exceeded by total weight of selected packages. This logistic problem is a direct analogon to packing of mountaineers knapsack.

– Ví dụ minh họa thứ 3 về tình hình kinh tế thực tế được (KP) nắm bắt được lấy từ hoạt động vận chuyển hàng hóa của hãng hàng không. Người điều phối của 1 hãng hàng không phải quyết định nên đáp ứng yêu cầu vận chuyển nào của khách hàng, tức là cách chất hàng lên 1 chiếc máy bay cụ thể. Quyết định của anh ta dựa trên danh sách các yêu cầu có chứa trọng lượng w_i của mọi kiện hàng & giá cước theo đơn vị trọng lượng được tính cho mỗi yêu cầu. Lưu ý: giá cước này không cố định mà phụ thuộc vào các thỏa thuận dài hạn cụ thể với từng khách hàng. Do đó, lợi nhuận p_i mà công ty kiếm được khi chấp nhận 1 yêu cầu & khi đặt kiện hàng tương ứng lên máy bay không tỷ lệ thuận với trọng lượng của kiện hàng. Đương nhiên, mỗi máy bay đều có sức chứa tối đa quy định c không được vượt quá tổng trọng lượng của các kiện hàng đã chọn. Vấn đề hậu cần này tương tự trực tiếp với việc đóng gói ba lô của người leo núi.

While previous examples all contain elements of "packing", one may also view (KP) as a "cutting" problem. Assume: a sawmill has to cut a log into shorter pieces. Pieces must however be cut into some predefined standard-lengths w_i, where each length has an associated selling price p_i. In order to maximize profit of log, sawmill can formulate problem as a (KP) where length of log defines capacity c.

– Trong khi các ví dụ trước đều chứa các yếu tố của "đóng gói", người ta cũng có thể xem (KP) là 1 vấn đề "cắt". Giả sử: 1 xưởng cưa phải cắt 1 khúc gỗ thành các đoạn ngắn hơn. Tuy nhiên, các đoạn gỗ phải được cắt thành 1 số chiều dài chuẩn được xác định trước w_i, trong đó mỗi chiều dài có giá bán liên quan là p_i. Để tối đa hóa lợi nhuận của khúc gỗ, xưởng cưa có thể xây dựng vấn đề như 1 (KP) trong đó chiều dài của khúc gỗ xác định công suất c.

An interesting example of knapsack problem from academia which may appeal to teachers & students was reported by Feuerman & Weiss [144]. They describe a test procedure from a college in Norwalk, Connecticut, where students may select a subset of given question. To be more precise, students receive n questions each with a certain "weight" indicating number of points that can be scored for that question with a total of e.g. 125 points. However, after exam all questions answered by students are graded by instructor who assigns points to teach answer. Then a subset of questions is selected to determine overall grade s.t. maximum number of reachable points for this subset is below a certain threshold, e.g., 100. To give students best possible marks subset should be chosen automatically s.t. scored points are as large as possible. This task is clearly equivalent to solving a knapsack problem with an item i for ith question with w_i representing reachable points & p_i actually scored points. Capacity c gives threshold for limit of points of selected questions.

– 1 ví dụ thú vị về bài toán ba lô từ học viện có thể hấp dẫn giáo viên & học sinh đã được Feuerman & Weiss [144] báo cáo. Họ mô tả 1 quy trình kiểm tra từ 1 trường cao đẳng ở Norwalk, Connecticut, trong đó học sinh có thể chọn 1 tập hợp con của câu hỏi cho sẵn. Để chính xác hơn, học sinh nhận được n câu hỏi, mỗi câu hỏi có 1 "trọng số" nhất định biểu thị số điểm có thể đạt được cho câu hỏi đó với tổng số điểm là 125 điểm. Tuy nhiên, sau khi thi, tất cả các câu hỏi do học sinh

trả lời đều được giáo viên chấm điểm, người này sẽ chỉ định điểm để dạy trả lời. Sau đó, 1 tập hợp con các câu hỏi được chọn để xác định điểm tổng thể s.t. số điểm tối đa có thể đạt được cho tập hợp con này thấp hơn 1 ngưỡng nhất định, ví dụ: 100. Để cho học sinh điểm tốt nhất có thể, tập hợp con nên được chọn tự động s.t. điểm ghi được càng lớn càng tốt. Nhiệm vụ này rõ ràng tương đương với việc giải bài toán ba lô với 1 mục i cho câu hỏi ith với w_i biểu thị điểm có thể đạt được & p_i điểm thực tế đã ghi được. Dung lượng c cung cấp ngưỡng giới hạn điểm của các câu hỏi đã chọn.

However, as it is frequently case with industrial applications, in practice several additional constraints, e.g. urgency & priority of requests, time windows for every request, packages with low weight but high volume, etc., have to be fulfilled. This leads to various extensions & variations of basic model (KP). Because this need for extension of basic knapsack model arose in many practical optimization problems, some of more general variants of (KP) have become standard problems of their own. Introduce several of them in following sect & deal with many others in later chaps of this book.

– Tuy nhiên, như thường thấy trong các ứng dụng công nghiệp, trong thực tế, 1 số ràng buộc bổ sung, ví dụ như tính cấp bách & mức độ ưu tiên của các yêu cầu, khung thời gian cho mọi yêu cầu, các gói có trọng lượng thấp nhưng khối lượng lớn, v.v., phải được đáp ứng. Điều này dẫn đến nhiều phần mở rộng & biến thể của mô hình cơ bản (KP). Vì nhu cầu mở rộng mô hình ba lô cơ bản này nảy sinh trong nhiều bài toán tối ưu hóa thực tế, 1 số biến thể tổng quát hơn của (KP) đã trở thành các bài toán chuẩn của riêng chúng. Giới thiệu 1 số trong số chúng trong phần sau & giải quyết nhiều bài toán khác trong các chương sau của cuốn sách này.

Beside these explicit occurrences of knapsack problems it should be noted: many solution methods of more complex problems employ knapsack problem (sometimes iteratively) as a subproblem. Therefore, a comprehensive study of knapsack problem carries many advantages for a wide range of mathematical models.

– Bên cạnh những trường hợp rõ ràng này của bài toán ba lô, cần lưu ý: nhiều phương pháp giải bài toán phức tạp hơn sử dụng bài toán ba lô (đôi khi lặp lại) như 1 bài toán con. Do đó, 1 nghiên cứu toàn diện về bài toán ba lô mang lại nhiều lợi thế cho nhiều mô hình toán học.

From a didactic & historic point of view it is worth mentioning: many techniques of combinatorial optimization & also of CS were introduced in context of, or in connection with knapsack problems. 1 of 1st optimization problems to be considered in development of NP-hardness (Sect. 1.5) was subset sum problem (Sect. 1.2). Other concepts e.g. approximation schemes, reduction algorithms & dynamic programming were established in their beginning based on or illustrated by knapsack problem.

– Theo quan điểm giáo khoa & lịch sử, điều đáng đề cập là: nhiều kỹ thuật tối ưu hóa tổ hợp & cũng như của CS đã được giới thiệu trong bối cảnh hoặc liên quan đến các bài toán ba lô. 1 trong những bài toán tối ưu hóa đầu tiên được xem xét trong quá trình phát triển độ khó NP (Phần 1.5) là bài toán tổng tập con (Phần 1.2). Các khái niệm khác ví dụ như các lược đồ xấp xỉ, thuật toán rút gọn & lập trình động đã được thiết lập ngay từ đầu dựa trên hoặc minh họa bằng bài toán ba lô.

Research in combinatorial optimization & operational research can be carried out either in a top-down or bottom-up fashion. In top-down approach, researchers develop solution methods for most difficult optimization problems like *traveling salesman problem, quadratic assignment problem* or *scheduling problem*. If developed methods work for these difficult problems, may assume: they will also work for a large variety of other problems. Opposite approach: develop new methods for most simple model, like e.g. *knapsack problem*, hoping: techniques can be generalized to more complex models. Since both approaches are *method developing*, they justify a considerable research effort for solving a relatively simple problem.

– Nghiên cứu về tối ưu hóa tổ hợp & nghiên cứu vận hành có thể được thực hiện theo cách từ trên xuống hoặc từ dưới lên. Theo cách tiếp cận từ trên xuống, các nhà nghiên cứu phát triển các phương pháp giải cho hầu hết các bài toán tối ưu hóa khó như *bài toán người bán hàng du lịch, bài toán gán bậc hai* hoặc *bài toán lập lịch*. Nếu các phương pháp đã phát triển có hiệu quả đối với các bài toán khó này, có thể cho rằng: chúng cũng sẽ có hiệu quả đối với nhiều bài toán khác. Cách tiếp cận ngược lại: phát triển các phương pháp mới cho hầu hết các mô hình đơn giản, chẳng hạn như *bài toán ba lô*, hy vọng: các kỹ thuật có thể được khái quát hóa thành các mô hình phức tạp hơn. Vì cả hai cách tiếp cận đều là *phương pháp đang phát triển*, chúng biện minh cho nỗ lực nghiên cứu đáng kể để giải quyết 1 bài toán tương đối đơn giản.

Skiena [437 reports an analysis of a quarter of a million requests to Stony Brook Algorithms Repository, to determine relative level of interest among 75 algorithmic problems. In this analysis it turns out: codes for knapsack problem are among top-20 of most most requested algorithms. When comparing interest to number of actual knapsack implementations, Skiena concludes: knapsack algorithms are 3rd most needed implementations. Research should not be driven by demand figures alone, but analysis indicates: knapsack problems occur in many real-life applications & solution of these problems is of vital interest both to industry & administration.

– Skiena [437 báo cáo phân tích 1 phần tư triệu yêu cầu gửi đến Stony Brook Algorithms Repository, để xác định mức độ quan tâm tương đối trong số 75 vấn đề thuật toán. Trong phân tích này, kết quả cho thấy: mã cho vấn đề knapsack nằm trong top 20 thuật toán được yêu cầu nhiều nhất. Khi so sánh mức độ quan tâm với số lượng triển khai knapsack thực tế, Skiena kết luận: thuật toán knapsack là triển khai cần thiết thứ 3. Nghiên cứu không nên chỉ dựa trên số liệu nhu cầu, nhưng phân tích chỉ ra rằng: vấn đề knapsack xảy ra trong nhiều ứng dụng thực tế & giải pháp của những vấn đề này có ý nghĩa sống còn đối với cả ngành & quản lý.

- 1.2. Variants & Extensions of Knapsack Problem. Consider again previous problem of cargo airline dispatcher. In a different setting profit made by accepting a package may be directly proportional to its weight. In this case optimal loading of plane is achieved by filling as much weight as possible into plane or equivalently setting $p_i = w_i$ in (KP). Resulting optimization problem is known as *subset sum problem* (SSP) because looking for a *subset* of values w_i with *sum* being as close as possible

to, but not exceeding given target value c.

$$(SSP)\ \max \sum_{i=1}^{n} w_i x_i \text{ subject to } \sum_{i=1}^{n} w_i x_i \leq c,\ x_i \in \{0,1\},\ \forall i \in [n].$$

– Xem xét lại bài toán trước của người điều phối hàng không vận chuyển hàng hóa. Trong 1 bối cảnh khác, lợi nhuận thu được khi chấp nhận 1 gói hàng có thể tỷ lệ thuận với trọng lượng của nó. Trong trường hợp này, tải trọng tối ưu của máy bay đạt được bằng cách chất càng nhiều trọng lượng càng tốt vào máy bay hoặc tương đương là đặt $p_i = w_i$ trong (KP). Bài toán tối ưu hóa kết quả được gọi là *subset sum problem* (SSP) vì tìm kiếm 1 *subset* các giá trị w_i với *sum* càng gần càng tốt nhưng không vượt quá giá trị mục tiêu c đã cho.

In original cargo problem described above it will frequently be case that not all packages are different from each other. In particular, in practice there may be given a number b_i of identical copies of each item to be transported. If either have to accept a request $\forall b_i$ packages or reject them all then we can generate an artificial request with weight $b_i w_i$ generating a profit of $b_i p_i >$ If possible to select also a subset of b_i items from a request we can either represent each individual package by a binary variable or more efficiently represent whole set of identical packages by an integer variable $x_i \geq 0$ indicating number of packages of this type which are put into plane. In this case number of variables is = number of different packages instead of total number of packages. This may decrease size of model considerably if numbers b_i are relatively large. Formally, constraint (1.3) in (KP) is replaced by (1.4)

$$0 \leq x_i \leq b_i,\ x_i \in \mathbb{N},\ \forall i \in [n].$$

Resulting problem is called *bounded knapsack problem* (BKP). Chap. 7 is devoted to (BKP). A special variant thereof is *unbounded knapsack problem* (UKP) (Chap. 8). Also known as *integer knapsack problem* where instead of a fixed number b_i a very large or an infinite amount of identical copies of each item is given. In this case, constraint (1.3) in (KP) is simply replaced by (1.5)

$$x_i \in \mathbb{N},\ \forall i \in [n].$$

Moving in a different direction, consider again above cargo problem & now take into account not only weight constraint but also limited space available to transport packages. For practical purposes only volume of packages is considered & not their different shapes.

– Trong bài toán vận chuyển hàng hóa ban đầu được mô tả ở trên, thường xảy ra trường hợp không phải tất cả các gói hàng đều khác nhau. Đặc biệt, trong thực tế, có thể đưa ra số lượng b_i bản sao giống hệt nhau của mỗi mặt hàng cần vận chuyển. Nếu phải chấp nhận yêu cầu $\forall b_i$ gói hàng hoặc từ chối tất cả thì chúng ta có thể tạo 1 yêu cầu nhân tạo với trọng số $b_i w_i$ tạo ra lợi nhuận là $b_i p_i >$ Nếu có thể chọn cả 1 tập hợp con của b_i mặt hàng từ 1 yêu cầu, chúng ta có thể biểu diễn từng gói hàng riêng lẻ bằng 1 biến nhị phân hoặc hiệu quả hơn là biểu diễn toàn bộ tập hợp các gói hàng giống hệt nhau bằng 1 biến số nguyên $x_i \geq 0$ biểu thị số lượng các gói hàng cùng loại được đưa vào mặt phẳng. Trong trường hợp này, số lượng biến là = số lượng các gói hàng khác nhau thay vì tổng số các gói hàng. Điều này có thể làm giảm đáng kể kích thước của mô hình nếu số lượng b_i tương đối lớn. Về mặt hình thức, ràng buộc (1.3) trong (KP) được thay thế bằng (1.4)

$$0 \leq x_i \leq b_i,\ x_i \in \mathbb{N},\ \forall i \in [n].$$

Bài toán kết quả được gọi là *bounded knapsack problem* (BKP). Chương 7 dành riêng cho (BKP). 1 biến thể đặc biệt của nó là *unbounded knapsack problem* (UKP) (Chương 8). Còn được gọi là *integer knapsack problem* trong đó thay vì 1 số cố định b_i, 1 số lượng rất lớn hoặc vô hạn các bản sao giống hệt nhau của mỗi mục được đưa ra. Trong trường hợp này, ràng buộc (1.3) trong (KP) chỉ được thay thế bằng (1.5)

$$x_i \in \mathbb{N},\ \forall i \in [n].$$

Di chuyển theo 1 hướng khác, hãy xem xét lại bài toán hàng hóa ở trên & bây giờ hãy tính đến không chỉ hạn chế về trọng lượng mà còn cả không gian hạn chế có sẵn để vận chuyển các gói hàng. Đối với mục đích thực tế, chỉ thể tích của các gói hàng được xem xét & không phải hình dạng khác nhau của chúng.

Denoting weight of every item by w_{1i} & its volume by w_{2i} & introducing weight capacity of plane as c_1 & upper bound on volume as c_2 we can formulate extended cargo problem by replacing constraint (1.2) in (KP) by 2 inequalities:

$$\sum_{i=1}^{n} w_{1i} x_i \leq c_1,\ \sum_{i=1}^{n} w_{2i} x_i \leq c_2, .$$

Obvious generalization of this approach, where d instead of 2 inequalities are introduced, yields *d-dimensional knapsack problem* or *multidimensional knapsack problem* (Chap. 9) formally defined by

$$(d-KP)\ \max \sum_{i=1}^{n} p_i x_i \text{ subject to } \sum_{j=1}^{n} w_{ij} x_j \leq c_i,\ \forall i \in [d],\ x_j \in \{0,1\},\ \forall j \in [n].$$

Another interesting variant of cargo problem arises from original version described above if consider a very busy flight route, e.g. Frankfurt–New York, which is flown by several planes everyday. In this case dispatcher has to decide on loading of a

number of planes in parallel, i.e., it has to be decided whether to accept a particular transportation request & in positive case on which plane to put corresponding package. Concepts of profit, weight, & capacity remain unchanged. This can be formulated by introducing a binary decision variable for every combination of a package with a plane. If there are n items on list of transportation requests & m planes available on this route we use mn binary variables x_{ij} for $i \in [m], j \in [n]$ with (1.6)

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is put into plane } i, \\ 0 & \text{otherwise.} \end{cases}$$

Mathematical programming formulation of this *multiple knapsack problem* (MKP) is given by

$$(MKP)\ \max \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \text{ subject to } \sum_{j=1}^{n} w_j x_{ij} \leq c_i,\ \forall i \in [m],\ \sum_{i=1}^{m} x_{ij} \leq 1,\ \forall j \in [n],\ x_{ij} \in \{0,1\},\ \forall i \in [m], \forall j \in [n].$$

p. 27+++

- 1.3. Single-Capacity vs. All-Capacities Problem.

- 2. Basic Algorithmic Concepts.
- 3. Advanced Algorithmic Concepts.
- 4. Subset Sum Problem.
- 5. Exact Solution of Knapsack Problem.
- 6. Approximation Algorithms for Knapsack Problem.
- 7. Bounded Knapsack Problem.
- 8. Unbounded Knapsack Problem.
- 9. Multidimensional Knapsack Problems.
- 10. Multiple Knapsack Problems.
- 11. Multiple-Choice Knapsack Problem.
- 12. Quadratic Knapsack Problem.
- 13. Other Knapsack Problems.
- 14. Stochastic Aspects of Knapsack Problems.
- 15. Some Selected Applications.
- Appendix A: Introduction to NP-Completeness of Knapsack Problems.

3 Scheduling Problems – Bài Toán Phân Công Thời Gian

3.1 AI-generated overview

Machine learning (ML) is used in job shop scheduling to create efficient solutions for the complex combinatorial optimization problem of assigning tasks to machines over time, often replacing traditional heuristic rules. Deep Reinforcement Learning (DRL) is a key technique, employing agents to learn optimal decision-making through trial & error by maximizing rewards, & Graph Neural Networks (GNNs) are used to represent the complex interdependencies in the scheduling problem as graphs. Other ML methods include deep neural networks (DNNs) for subproblem prediction & the development of modular libraries like JobShopLib to facilitate rapid experimentation & advancement in the field.

How ML is Applied.

1. Reinforcement Learning (RL): Agents are trained to learn scheduling policies by interacting with the scheduling environment, receiving rewards for good decisions (e.g., reducing makespan) & penalties for poor ones.
2. Graph Neural Networks (GNNs): The scheduling problem is often represented as a graph, where nodes represent jobs & machines & edges represent relationships & dependencies. GNNs can then process this graphical information to learn efficient scheduling rules.
3. Deep Neural Networks (DNNs): DNNs are used to predict good-enough solutions for subproblems within a larger scheduling framework, such as in a Surrogate Lagrangian Relaxation (SLR) method, which helps manage large-scale problem difficulties.
4. Imitation Learning: In some cases, ML models are trained to mimic the behavior of human-designed dispatching rules or well-performing existing dispatchers.

Key Challenges & Solutions.

1. Generalization & Scalability: ML models need to generalize to diverse problem instances & large-scale job shops. Methods like using heterogeneous graphs & intelligent decomposition strategies are used to address these challenges.

2. Representation: Creating effective state & action representations is crucial for RL agents. Researchers use techniques like disjunctive graphs & heterogeneous graphs to capture complex relationships.

3. Action Masking: To improve efficiency, invalid actions (e.g., trying to schedule a completed operation) are masked, narrowing the search space for the learning agent.

4. Uncertainty: Real-world scenarios involve task durations that are not precisely known. ML approaches are being developed to learn robust schedules that perform well under these uncertain conditions.

Benefits of ML in Job Shop Scheduling.

1. Automation: ML reduces the reliance on human experts for creating schedules.

2. Speed: ML models can generate solutions much faster than traditional methods, crucial for real-time adjustments.

3. Adaptability: ML can learn to adapt to dynamic changes in the manufacturing environment.

4. Performance: ML-based methods, especially those using DRL & GNNs, have shown superior performance compared to traditional dispatching rules & other state-of-the-art algorithms.

3.2 Serge Kruk. Practical Python AI Projects: Mathematical Models of Optimization Problems with Google OR-Tools. 2018

- 1. Introduction.
  - 1.1. What Is This Book About? AI is a wide field covering diverse techniques, objectives, & measures of success. 1 branch is concerned with finding provably optimal solutions to some well-defined problems. This book is an introduction to art & science of implementing *mathematical models of optimization problems*.

    – Trí tuệ nhân tạo (AI) là 1 lĩnh vực rộng lớn, bao gồm nhiều kỹ thuật, mục tiêu & thước đo thành công khác nhau. 1 nhánh liên quan đến việc tìm kiếm các giải pháp tối ưu có thể chứng minh được cho 1 số vấn đề được xác định rõ ràng. Cuốn sách này là phần giới thiệu về nghệ thuật & khoa học triển khai *các mô hình toán học của các bài toán tối ưu hóa*.

    An optimization problem is almost any problem that is, or can be, formulated as a question starting with "What is the best . . .?" E.g.,

    * What is best route to get from home to work?
    * What is best way to produce cars to maximize profit?
    * What is best way to carry groceries home: paper or plastic?
    * Which is best school for my kid?
    * Which is best fuel to use in rocket boosters?
    * What is best placement of transistors on a chip?
    * What is best NBA schedule?

    These questions are rather vague & can be interpreted in a multitude of ways. Consider 1st: by "best" do we mean fastest, shortest, most pleasant to ride, least bumpy, or least fuel-guzzling? Besides, question is incomplete. Are we walking, riding, driving, or snowboarding? Are we alone or accompanied by a screaming toddler?

    – Bài toán tối ưu hóa gần như là bất kỳ bài toán nào được, hoặc có thể được, xây dựng dưới dạng 1 câu hỏi bắt đầu bằng "Đường nào tốt nhất?". Ví dụ:

    * Tuyến đường nào tốt nhất để đi từ nhà đến nơi làm việc?
    * Cách tốt nhất để sản xuất ô tô để tối đa hóa lợi nhuận là gì?
    * Cách tốt nhất để mang đồ tạp hóa về nhà: giấy hay nhựa?
    * Trường nào là tốt nhất cho con tôi?
    * Nhiên liệu nào tốt nhất để sử dụng trong tên lửa đẩy?
    * Vị trí đặt bóng bán dẫn trên chip tốt nhất là gì?
    * Lịch thi đấu NBA tốt nhất là gì?

    Những câu hỏi này khá mơ hồ & có thể được diễn giải theo nhiều cách. Hãy xem xét điều thứ nhất: "tốt nhất" ở đây có nghĩa là nhanh nhất, ngắn nhất, dễ đi nhất, ít xóc nhất hay ít tốn nhiên liệu nhất? Hơn nữa, câu hỏi vẫn chưa đầy đủ. Chúng ta đang đi bộ, cưỡi ngựa, lái xe hay trượt tuyết? Chúng ta đi 1 mình hay có 1 đứa trẻ đang la hét đi cùng?

    To help us formulate solutions to optimize problems, optimizers [I use term "optimizers" to name mathematicians, theoreticians, & practitioners, who, since 1]950s, have worked in fields of linear programming (LP) & integer programming (IP). There are others who could make valid claims to moniker, chiefly among them researchers in constraint programming, but my focus will be mostly in LP & IP models, hence my restricted definition.] have established a frame into which we mould

questions; it's called a model. Most crucial aspect of a model: it has an objective & it has constraints. Roughly, objective is what we want & constraints are obstacles in our way. If can reformulate question to clearly identify both objective & constraints, we are closer to a model.

– Để giúp chúng ta xây dựng các giải pháp tối ưu hóa các vấn đề, những người tối ưu hóa [tôi sử dụng thuật ngữ "người tối ưu hóa" để đặt tên cho các nhà toán học, nhà lý thuyết, & người thực hành, những người, từ những năm 1950, đã làm việc trong các lĩnh vực lập trình tuyến tính (LP) & lập trình số nguyên (IP). Có những người khác có thể đưa ra những tuyên bố hợp lệ về biệt danh này, chủ yếu trong số họ là các nhà nghiên cứu về lập trình ràng buộc, nhưng trọng tâm của tôi sẽ chủ yếu là các mô hình LP & IP, do đó định nghĩa của tôi bị hạn chế.] đã thiết lập 1 khuôn khổ mà chúng ta định hình các câu hỏi; nó được gọi là 1 mô hình. Khía cạnh quan trọng nhất của 1 mô hình: nó có mục tiêu & nó có các ràng buộc. Nói 1 cách đại khái, mục tiêu là những gì chúng ta muốn & ràng buộc là những trở ngại trên con đường của chúng ta. Nếu có thể xây dựng lại câu hỏi để xác định rõ ràng cả mục tiêu & ràng buộc, chúng ta sẽ tiến gần hơn đến 1 mô hình.

Consider in more detail "best route" problem but with an eye to clarify objective & constraints. Could formulate it as: Given a map of city, my home address, & address of daycare of my 2-year-old son, what is best route to take on my bike to bring him to daycare as fast as possible? Goal: find among all solutions that satisfy requirements (i.e., paths following either streets or bike lanes, also known as constraints) 1 path that minimizes time it takes to get there (objective).

– Hãy xem xét bài toán "tuyến đường tốt nhất" 1 cách chi tiết hơn, nhưng cần làm rõ các ràng buộc & mục tiêu. Có thể diễn đạt bài toán như sau: Cho bản đồ thành phố, địa chỉ nhà tôi, & địa chỉ nhà trẻ của con trai 2 tuổi, đâu là tuyến đường tốt nhất để tôi đạp xe đưa con đến nhà trẻ nhanh nhất có thể? Mục tiêu: tìm trong số tất cả các giải pháp thỏa mãn các yêu cầu (ví dụ: đường đi theo đường phố hoặc làn đường dành cho xe đạp, còn được gọi là ràng buộc) 1 tuyến đường giúp giảm thiểu thời gian đến đích (mục tiêu).

Objectives are always quantities we want to maximize or minimize (time, distance, money, surface area, etc.), although you will see examples where we want to maximize something & minimize something else; this is easily accommodated. Sometimes there are no objectives. Say: problem is 1 of feasibility (i.e., we are looking for any solution satisfying requirements). From point of view of modeler, difference is minimal. Especially since, in most practical cases, a feasibility model is usually a 1st step. After noticing a solution, one usually wants to optimize something & model is modified to include an objective function.

– Mục tiêu luôn là những đại lượng chúng ta muốn tối đa hóa hoặc tối thiểu hóa (thời gian, khoảng cách, tiền bạc, diện tích bề mặt, v.v.), mặc dù bạn sẽ thấy những ví dụ trong đó chúng ta muốn tối đa hóa 1 thứ gì đó & tối thiểu hóa 1 thứ khác; điều này dễ dàng được đáp ứng. Đôi khi không có mục tiêu nào cả. Ví dụ: bài toán có mức độ khả thi 1 (tức là chúng ta đang tìm kiếm bất kỳ giải pháp nào thỏa mãn các yêu cầu). Theo quan điểm của người lập mô hình, sự khác biệt là rất nhỏ. Đặc biệt là vì, trong hầu hết các trường hợp thực tế, mô hình khả thi thường là bước đầu tiên. Sau khi nhận thấy 1 giải pháp, người ta thường muốn tối ưu hóa 1 thứ gì đó & mô hình được điều chỉnh để bao gồm 1 hàm mục tiêu.

- 1.2. Features of Text. As this text is an intro, do not expect reader to be already well versed in art of modeling. Start at beginning, assuming only reader understands definition of a variable (both in mathematical sense & in programming sense), an equation, an inequality, & a function. Also assume: reader knows some programming language, preferably Python, although knowing any other imperative language is enough to be able to read Python code displayed in text.

  – Vì đây là phần giới thiệu, đừng kỳ vọng người đọc đã thành thạo về nghệ thuật mô hình hóa. Hãy bắt đầu từ đầu, giả định rằng chỉ người đọc hiểu định nghĩa của 1 biến (cả về mặt toán học & về mặt lập trình), 1 phương trình, 1 bất đẳng thức, & 1 hàm. Cũng giả định rằng: người đọc biết 1 ngôn ngữ lập trình nào đó, tốt nhất là Python, mặc dù biết bất kỳ ngôn ngữ lập trình bắt buộc nào khác cũng đủ để đọc được mã Python hiển thị trong văn bản.

  Note code in this book is an essential component. To get full value, reader must, slowly & attentively, read code. This book is not a text of recipes described from a birds-eye view, using mathematical notation, with all nitty-gritty details "left as an exercise for reader". This is implemented, functional, tested, optimization code that reader can use & moreover is encouraged to modify to fully understand. Mathematics in book has been reviewed by mathematicians, like any mathematical paper. But code has been subjected to a much more stringent set of reviewers with names Intel, AMD, Motorola, & IBM. [My doctoral advisor used to say "There are error-free mathematical papers." But we only have found an existence proof of that theorem. I will not claim: code is error-free, but I am certain: it has fewer errors than any mathematical paper I ever wrote.]

  – Lưu ý rằng mã trong cuốn sách này là 1 thành phần thiết yếu. Để có được giá trị trọn vẹn, người đọc phải đọc mã 1 cách chậm rãi & chăm chú. Cuốn sách này không phải là 1 văn bản công thức được mô tả từ góc nhìn tổng quan, sử dụng ký hiệu toán học, với tất cả các chi tiết cụ thể "được để lại như 1 bài tập cho người đọc". Đây là mã được triển khai, hoạt động, đã được kiểm tra & tối ưu hóa mà người đọc có thể sử dụng & hơn nữa được khuyến khích sửa đổi để hiểu đầy đủ. Toán học trong sách đã được các nhà toán học đánh giá, giống như bất kỳ bài báo toán học nào. Nhưng mã đã được kiểm tra bởi 1 nhóm các nhà đánh giá nghiêm ngặt hơn nhiều, những người có tên tuổi như Intel, AMD, Motorola, & IBM. [Giáo sư hướng dẫn tiến sĩ của tôi thường nói "Có những bài báo toán học không có lỗi." Nhưng chúng tôi chỉ mới tìm thấy bằng chứng tồn tại của định lý đó. Tôi sẽ không khẳng định: mã không có lỗi, nhưng tôi chắc chắn: nó có ít lỗi hơn bất kỳ bài báo toán học nào tôi từng viết.]

  Book is fruit of decades of consulting & years teaching both an introductory modeling class (MOR242 Intro to Operation Research Models) & a graduate class (APM568 Mathematical Modeling in Industry) at Oakland University. Start at undergraduate level & proceed up to graduate level in terms of modeling itself, without delving much into attendant theory.

  * Every model is expressed in Python using Google OR-Tools & can be executed as stated. In fact, code presented in book is automatically extracted, executed, & output inserted into text without manual intervention; even graphs are produced automatically (thanks to Emacs & org-mode).

* My intention: help reader become a proficient modeler, not a theoretician. Therefore, little of fascinating mathematical theory related to optimization is covered. It is nevertheless used profitably to create simple yet efficient models.
* Associated web site provides all code presented in book along with a random generator for many of problems & variations. Author uses this as a personalized homework generator. It can also be used as a self-guided learning tool. https://github.com/sgkruk/Apress-AI

– Cuốn sách là thành quả của nhiều thập kỷ tư vấn & nhiều năm giảng dạy cho cả lớp mô hình hóa cơ bản (MOR242 Giới thiệu về Mô hình Nghiên cứu Vận hành) & lớp sau đại học (APM568 Mô hình Toán học trong Công nghiệp) tại Đại học Oakland. Bắt đầu từ bậc đại học & tiến lên bậc sau đại học về bản chất của mô hình hóa, mà không cần đào sâu vào lý thuyết liên quan.

* Mọi mô hình đều được biểu diễn bằng Python sử dụng Google OR-Tools & có thể được thực thi như đã nêu. Trên thực tế, mã được trình bày trong sách được tự động trích xuất, thực thi, & chèn đầu ra vào văn bản mà không cần can thiệp thủ công; ngay cả đồ thị cũng được tạo tự động (nhờ Emacs & org-mode).
* Mục đích của tôi: giúp người đọc trở thành 1 nhà mô hình hóa thành thạo, chứ không phải 1 nhà lý thuyết. Do đó, sách không đề cập nhiều đến lý thuyết toán học hấp dẫn liên quan đến tối ưu hóa. Tuy nhiên, sách vẫn được sử dụng hiệu quả để tạo ra các mô hình đơn giản nhưng hiệu quả.
* Trang web liên kết cung cấp tất cả mã được trình bày trong sách cùng với 1 trình tạo ngẫu nhiên cho nhiều bài toán & biến thể. Tác giả sử dụng trang web này như 1 trình tạo bài tập về nhà được cá nhân hóa. Nó cũng có thể được sử dụng như 1 công cụ tự học. https://github.com/sgkruk/Apress-AI
* 1.2.1. Running Models. There is danger in describing in too much detail installations instructions because software tends to change more often than this text will change. E.g., when I started with Google's OR-Tools, it was hosted on Google Code repository; now it is on GitHub. Nevertheless, here a few pointers. All code presented here has been tested with
  · Python 3 (currently 3.7), although models will work on Python 2
  · OR-Tools 6.6

  Page https://developers.google.com/optimization offers installation instructions for most OSs. Fastest & most painless way is `pip install --upgrade ortools`. Once OR-Tools are installed, software of this text can be downloaded most easily by cloning GitHub repo by gitclonehttps://github.com/sgkruk/Apress-AI.git where reader will find a Makefile testing almost all models detailed in text. Reader only has to issue a make to test that installation was completed successfully.
  – Việc mô tả hướng dẫn cài đặt quá chi tiết sẽ rất nguy hiểm vì phần mềm thường thay đổi thường xuyên hơn so với nội dung văn bản này. Ví dụ: khi tôi bắt đầu với OR-Tools của Google, nó được lưu trữ trên kho lưu trữ Google Code; giờ thì nó đã có trên GitHub. Tuy nhiên, sau đây là 1 vài điểm cần lưu ý. Tất cả mã được trình bày ở đây đã được kiểm tra với
  · Python 3 (hiện tại là 3.7), mặc dù các mô hình sẽ hoạt động trên Python 2
  · OR-Tools 6.6

  Trang https://developers.google.com/optimization cung cấp hướng dẫn cài đặt cho hầu hết các hệ điều hành. Cách nhanh nhất & dễ dàng nhất là `pip install --upgrade ortools`. Sau khi OR-Tools được cài đặt, phần mềm trong văn bản này có thể được tải xuống dễ dàng nhất bằng cách sao chép kho lưu trữ GitHub bằng gitclonehttps://github.com/sgkruk/Apress-AI.git, tại đó, người đọc sẽ tìm thấy 1 Makefile kiểm tra hầu hết tất cả các mô hình được mô tả chi tiết trong văn bản. Người đọc chỉ cần đưa ra lệnh make để kiểm tra xem quá trình cài đặt đã hoàn tất thành công hay chưa.
  Code of each sect of book is separated into 2 parts: a model proper, shown in text, & a main driver to illustrate how to call model with some data. E.g., chap corresponding to set cover has a file named `set_cover.py` with model & a file named `test_set_cover.py` which will create a random instance, run model on it, & display result. Armed with these examples, reader should be able to modify to suit his needs. Important to understand mainline is in `test_set_cover.py` & that file needs to be executed.
  – Mã của mỗi phần trong sách được chia thành 2 phần: 1 mô hình riêng, được hiển thị trong văn bản, & 1 trình điều khiển chính để minh họa cách gọi mô hình với 1 số dữ liệu. Ví dụ: chương tương ứng với tập hợp bìa có 1 tệp tên là `set_cover.py` với mô hình & 1 tệp tên là `test_set_cover.py` sẽ tạo 1 phiên bản ngẫu nhiên, chạy mô hình trên đó, & hiển thị kết quả. Với những ví dụ này, người đọc sẽ có thể sửa đổi cho phù hợp với nhu cầu của mình. Điều quan trọng cần hiểu là dòng chính nằm trong `test_set_cover.py` & tệp đó cần được thực thi.
* 1.2.2. A Note on Notation. Throughout book, describe algebraic models. These models can be represented in a number of ways. I will use 2. Sketch each model using common mathematical notation typeset with TEX in math mode. Then express compete, detailed model in executable Python code. Reader should have no problem setting equivalence between formulations. Table 1.1: Equivalence of Expression in Math & Python Modes. illustrates some of equivalencies. Summation: $\sum_{i=0}^{9} x_i$: `sum(x[i] for i in range(10))`, Set Definition: $\{i^2; i \in [0, 1, \ldots, 9]\}$: `[i**2 for i in range(10)]`.
  – Trong suốt cuốn sách, hãy mô tả các mô hình đại số. Các mô hình này có thể được biểu diễn theo nhiều cách. Tôi sẽ sử dụng 2. Phác thảo từng mô hình bằng cách sử dụng bộ ký hiệu toán học phổ biến với TEX ở chế độ toán học. Sau đó, biểu diễn mô hình chi tiết, hoàn chỉnh bằng mã Python có thể thực thi. Người đọc sẽ không gặp khó khăn khi thiết lập tính tương đương giữa các công thức. Bảng 1.1: Tính tương đương của biểu thức trong các chế độ toán học & Python. minh họa 1 số tính tương đương. Tổng: $\sum_{i=0}^{9} x_i$: `sum(x[i] for i in range(10))`, Định nghĩa tập hợp: $\{i^2; i \in [0, 1, \ldots, 9]\}$: `[i**2 for i in range(10)]`.

◦ 1.3. Getting Our Feet Wet: Amphibian Coexistence. Simplest problems are similar to those 1st encountered in high school: dreaded word problems. They are algebraic in nature; i.e., they can be formulated & sometimes solved using simple tools

of elementary linear algebra. Consider here 1 such problem to illustrate approach to modeling & define some fundamental concepts.

– Làm quen với môi trường: Sự chung sống của lưỡng cư. Những bài toán đơn giản nhất cũng tương tự như những bài toán đầu tiên gặp ở trường trung học: những bài toán lời văn đáng sợ. Chúng mang bản chất đại số; tức là, chúng có thể được xây dựng & đôi khi được giải bằng các công cụ đơn giản của đại số tuyến tính sơ cấp. Hãy xem xét 1 bài toán như vậy để minh họa cách tiếp cận mô hình hóa & định nghĩa 1 số khái niệm cơ bản.

A zoo biologist will place 3 species of amphibians (a toad, a salamander, & a caecilian) in an aquarium where they will feed on 3 different small preys: worms, crickets, & flies. Each day 1500 worms, 3000 crickets, & 5500 flies will be placed in aquarium. Each amphibian consumes a certain number of preys per day. Table 1.2: Number of Preys Consumed by Each Species of Amphibian summarizes relevant data.

– 1 nhà sinh vật học ở sở thú sẽ thả 3 loài lưỡng cư (một con cóc, 1 con kỳ nhông, & 1 con giun đất) vào 1 bể cá, nơi chúng sẽ ăn 3 loại mồi nhỏ khác nhau: giun, dế, & ruồi. Mỗi ngày, 1500 con giun, 3000 con dế, & 5500 con ruồi sẽ được thả vào bể. Mỗi loài lưỡng cư tiêu thụ 1 lượng mồi nhất định mỗi ngày. Bảng 1.2: Số lượng con mồi mà mỗi loài lưỡng cư tiêu thụ tóm tắt dữ liệu liên quan.

Biologist wants to know how many amphibians, up to 1000 of each species, can coexist in aquarium assuming that food is only relevant constraint. How do we model this problem? All optimization & feasibility problems in this book are modeled using a 3-step approach. Expand on this approach as encounter problems on increasing complexity, but fundamental 3 steps remain cornerstone of a good model.

– Nhà sinh vật học muốn biết có bao nhiêu loài lưỡng cư, tối đa 1000 cá thể mỗi loài, có thể cùng tồn tại trong bể cá, giả định rằng thức ăn chỉ là yếu tố hạn chế quan trọng. Làm thế nào để mô hình hóa bài toán này? Tất cả các bài toán tối ưu hóa & khả thi trong cuốn sách này đều được mô hình hóa bằng phương pháp 3 bước. Hãy mở rộng phương pháp này khi gặp phải các bài toán có độ phức tạp tăng dần, nhưng 3 bước cơ bản vẫn là nền tảng của 1 mô hình tốt.

1. **Identify question to answer.** This identification should take form of a precise sentence involving either counting or valuating 1 or more objects. In this case, how many amphibians each species can coexist in aquarium? Notice "How many amphibians?" would be not precise enough because not interested in total count, but rather in count of each species. Formulating a precise question is often hardest part. Once we have this precise equation, assign a variable to each of objects to count. Use x_0, x_1, x_2. These are traditionally known as *decision variables*. Expression is a misnomer in our 1st example but reflects origins of optimization problems in logistics where decision variables were indeed representative of quantities under control of modeler & mapped to planning decisions.

   – **Xác định câu hỏi cần trả lời.** Việc xác định này phải ở dạng 1 câu chính xác bao gồm việc đếm hoặc định giá 1 hoặc nhiều đối tượng. Trong trường hợp này, mỗi loài có thể cùng tồn tại bao nhiêu loài lưỡng cư trong bể cá? Lưu ý rằng "Có bao nhiêu loài lưỡng cư?" sẽ không đủ chính xác vì không quan tâm đến tổng số lượng mà là số lượng của từng loài. Việc xây dựng 1 câu hỏi chính xác thường là phần khó nhất. Khi chúng ta có phương trình chính xác này, hãy gán 1 biến cho mỗi đối tượng để đếm. Sử dụng x_0, x_1, x_2. Chúng thường được gọi là *biến quyết định*. Biểu thức là 1 cách gọi sai trong ví dụ đầu tiên của chúng ta nhưng phản ánh nguồn gốc của các vấn đề tối ưu hóa trong hậu cần, trong đó các biến quyết định thực sự đại diện cho số lượng dưới sự kiểm soát của người lập mô hình & được ánh xạ tới các quyết định lập kế hoạch.

2. **Identify all requirements & translate them into constraints.** Constraints, as see throughout book, can take on a multitude of forms. In this simple form, they are algebraic, linear inequalities. Often best to write down each requirement in a precise sentence before translating it into a constraint. For coexistence case, requirements, in words, are

   * All amphibians combined consume 1500 worms.
   * All amphibians combined consume 3000 crickets.
   * All amphibians combined consume 5000 flies.

   Note a statement starting with "The amount of . . ." may not be precise enough. In our simple case, there are no specified u nits but there could be. E.g., amount consumed could be stated in grams while availability is in kilograms. This happen often & is cause of many a model going awry.

   – **Xác định tất cả các yêu cầu & chuyển chúng thành các ràng buộc.** Các ràng buộc, như được thấy trong toàn bộ sách, có thể có nhiều dạng. Ở dạng đơn giản này, chúng là các bất đẳng thức đại số tuyến tính. Thông thường, tốt nhất là viết từng yêu cầu thành 1 câu chính xác trước khi chuyển nó thành 1 ràng buộc. Đối với trường hợp cùng tồn tại, các yêu cầu, được diễn đạt bằng lời, là

   * Tổng số lưỡng cư tiêu thụ 1500 con giun.
   * Tổng số lưỡng cư tiêu thụ 3000 con dế.
   * Tổng số lưỡng cư tiêu thụ 5000 con ruồi.

   Lưu ý rằng 1 câu lệnh bắt đầu bằng "Lượng . . ." có thể không đủ chính xác. Trong trường hợp đơn giản của chúng ta, không có đơn vị cụ thể nào được chỉ định nhưng có thể có. Ví dụ: lượng tiêu thụ có thể được tính bằng gam trong khi lượng khả dụng được tính bằng kilôgam. Điều này thường xảy ra & là nguyên nhân khiến nhiều mô hình bị sai lệch.

   Yet, even with our seemingly precise statements, there is an ambiguity left to consider. It is 1 of main contributions of a good modeler to highlight ambiguity & clarify problem statements. Here, do we mean: amphibians will consume exactly amounts stated, or that they will consume at most amounts stated? [This seemingly trivial change from "exactly equal" to "at most" represents > 2000 years of mathematical development in solution techniques. Have known how to solve "equal" form since ancient Babylonians (though it is known today as "Gaussian elimination") & teach it in high school, but only

discovered how to solve "at most" form in 20th century.] We will assume: "at most" is proper form of requirement, both because it is more interesting &, in a sense, subsumes "equal" question. Then translate these requirements into algebraic constraints based on our decision variables.

– Tuy nhiên, ngay cả với những tuyên bố có vẻ chính xác của chúng ta, vẫn còn 1 sự mơ hồ cần xem xét. 1 trong những đóng góp chính của 1 nhà mô hình hóa giỏi là làm nổi bật sự mơ hồ & làm rõ các phát biểu vấn đề. Ở đây, ý chúng ta là: lưỡng cư sẽ tiêu thụ chính xác lượng đã nêu, hay chúng sẽ tiêu thụ tối đa lượng đã nêu? [Sự thay đổi có vẻ tầm thường này từ "chính xác bằng" thành "tối đa" thể hiện hơn 2000 năm phát triển toán học trong các kỹ thuật giải toán. Chúng ta đã biết cách giải dạng "bằng" từ thời người Babylon cổ đại (mặc dù ngày nay được gọi là "phép loại trừ Gauss") & dạy nó ở trường trung học, nhưng chỉ phát hiện ra cách giải dạng "tối đa" vào thế kỷ 20.] Chúng ta sẽ giả định: "tối đa" là dạng yêu cầu thích hợp, 1 phần vì nó thú vị hơn &, theo 1 nghĩa nào đó, bao hàm câu hỏi "bằng". Sau đó, chuyển các yêu cầu này thành các ràng buộc đại số dựa trên các biến quyết định của chúng ta.

Consider worms. Toads eat 2 per day. Salamanders & caecilians each eat 1. Since we decided on x_0 toads, x_1 salamanders, & x_2 caecilians, total number of worms consumed will be bounded by following inequality $2x_0 + x_1 + x_2 \leq 1500$. Had we decided that "equal to" was proper constraint, we would replace inequality by an equality. Consider now crickets. Toads consume 1 per day while salamanders consume 3 & caecilians consume 2. They will collectively consume $x_0 + 3x_1 + 2x_2$ & obtain constraint $x_0 + 3x_1 + 2x_2 \leq 3000$. Constraint on flies is obtained similarly to produce $x_0 + 2x_1 + 3x_2 \leq 5000$.

– Hãy xem xét loài giun. Cóc ăn 2 con mỗi ngày. Kỳ nhông & giun đất mỗi con ăn 1 con. Vì chúng ta đã quyết định x_0 cóc, x_1 kỳ nhông, & x_2 giun đất, tổng số giun tiêu thụ sẽ bị giới hạn bởi bất đẳng thức sau $2x_0 + x_1 + x_2 \leq 1500$. Nếu chúng ta quyết định rằng "bằng" là ràng buộc đúng, chúng ta sẽ thay thế bất đẳng thức bằng 1 đẳng thức. Bây giờ hãy xem xét loài dế. Cóc tiêu thụ 1 con mỗi ngày trong khi kỳ nhông tiêu thụ 3 & giun đất tiêu thụ 2 con. Chúng sẽ tiêu thụ chung $x_0 + 3x_1 + 2x_2$ & thu được ràng buộc $x_0 + 3x_1 + 2x_2 \leq 3000$. Ràng buộc đối với ruồi cũng thu được tương tự để tạo ra $x_0 + 2x_1 + 3x_2 \leq 5000$.

3. **Identify objective to optimize.** Objective is, in case of an optimization problem, what we want to maximize (or minimize). In case of a feasibility problem, there is no objective, but in practice, most feasibility problems are really optimization problems that have been incompletely formulated. Since problem is stated as "How many amphibians of each species can coexist?", a possible, even likely, reading: want maximum number of amphibians. (Minimum number is 0 & is an example of uninteresting trivial solution.) In terms of our decision variables, want to maximize sum & obtain $\max x_0 + x_1 + x_2$.

   – **Xác định mục tiêu để tối ưu hóa.** Trong trường hợp bài toán tối ưu hóa, mục tiêu là những gì chúng ta muốn tối đa hóa (hoặc tối thiểu hóa). Trong trường hợp bài toán khả thi, không có mục tiêu, nhưng trên thực tế, hầu hết các bài toán khả thi thực chất là các bài toán tối ưu hóa chưa được xây dựng đầy đủ. Vì bài toán được phát biểu là "Có bao nhiêu loài lưỡng cư của mỗi loài có thể cùng tồn tại?", 1 cách đọc khả thi, thậm chí có thể xảy ra: muốn số lượng lưỡng cư tối đa. (Số lượng tối thiểu là 0 & là 1 ví dụ về giải pháp tầm thường không thú vị.) Về mặt các biến quyết định của chúng ta, muốn tối đa hóa tổng & thu được $\max x_0 + x_1 + x_2$.

At this point we have a model! Not the model, but a model: a simple, clear, & precise algebraic model that has a solution, one that answers our original question. Since we are not mere theoreticians uninterested in practical applications, our next step: solve model. As we will do for every model in this book, need to translate mathematical expressions above into a form digestible by 1 of many solvers available.

– Đến đây, chúng ta đã có 1 mô hình! Không phải mô hình, mà là 1 mô hình: 1 mô hình đại số đơn giản, rõ ràng, & chính xác, có lời giải, 1 lời giải trả lời cho câu hỏi ban đầu của chúng ta. Vì chúng ta không chỉ là những nhà lý thuyết không quan tâm đến ứng dụng thực tế, bước tiếp theo của chúng ta: giải mô hình. Như chúng ta sẽ làm với mọi mô hình trong sách này, cần phải dịch các biểu thức toán học ở trên sang dạng dễ hiểu hơn bởi 1 trong số nhiều trình giải có sẵn.

Over years, optimizers have developed a number of specialized modeling languages & solvers. A short list of better-known ones:

* Modeling languages: AMPL, GAMS, GMPL (MathProg), Minizinc, OPL, ZIMPL
* Solvers: CBC, CLP, CPLEX, ECLiPSe, Gecode, GLOP, GLPK, Gurobi, SCIP

Should maintain a distinction between *modeling languages*, formal constructions with specific vocabulary & grammars, & *solvers*, software packages that can read in models expressed in certain languages & write out solutions, although in some cases this distinction is blurry.

– Qua nhiều năm, các nhà tối ưu hóa đã phát triển 1 số ngôn ngữ mô hình hóa chuyên biệt & trình giải. Danh sách ngắn gọn những ngôn ngữ nổi tiếng hơn:

* Ngôn ngữ mô hình hóa: AMPL, GAMS, GMPL (MathProg), Minizinc, OPL, ZIMPL
* Trình giải: CBC, CLP, CPLEX, ECLiPSe, Gecode, GLOP, GLPK, Gurobi, SCIP

Nên duy trì sự phân biệt giữa *ngôn ngữ mô hình hóa*, các cấu trúc hình thức với từ vựng & ngữ pháp cụ thể, & *trình giải*, các gói phần mềm có thể đọc các mô hình được thể hiện bằng 1 số ngôn ngữ nhất định & viết ra các lời giải, mặc dù trong 1 số trường hợp, sự phân biệt này không rõ ràng.

As a modeler, one creates a model (in language X) which is then fed to a solver (solver Y). This can happen because solver Y knows how to parse language X or because there is a translator between language X & another language, say Z, which solver understands. This, over years, has been cause of much irritation ("What? You mean that I have to rewrite my model to use your solver?").

– Với tư cách là người lập mô hình, người ta tạo ra 1 mô hình (bằng ngôn ngữ X) sau đó đưa vào 1 trình giải (trình giải Y). Điều này có thể xảy ra vì trình giải Y biết cách phân tích cú pháp ngôn ngữ X hoặc vì có 1 trình biên dịch giữa ngôn ngữ X & 1 ngôn ngữ khác, chẳng hạn như Z, mà trình giải hiểu được. Điều này, trong nhiều năm, đã gây ra nhiều khó chịu ("Cái gì? Ý anh là tôi phải viết lại mô hình của mình để sử dụng trình giải của anh à?").

To make matters worse, these languages & solvers are not equivalent. Each has its strengths & weaknesses, its areas of specialization. After years of writing models in all languages above & then some, my preference today is to eschew specialized languages & to use a general-purpose programming language, e.g., Python, along with a library interfacing with multiple solvers. Throughout this book use Google's Operation Research Tools (OR-Tools), a very well-structure & easy-to-use library.

– Tệ hơn nữa, các ngôn ngữ & trình giải này không tương đương nhau. Mỗi ngôn ngữ đều có điểm mạnh & điểm yếu, cũng như lĩnh vực chuyên môn riêng. Sau nhiều năm viết mô hình bằng tất cả các ngôn ngữ trên & & 1 số ngôn ngữ khác, hiện tại tôi muốn tránh xa các ngôn ngữ chuyên ngành & để sử dụng 1 ngôn ngữ lập trình đa năng, ví dụ như Python, cùng với 1 thư viện giao tiếp với nhiều trình giải. Trong suốt cuốn sách này, tôi sử dụng Công cụ Nghiên cứu Hoạt động (OR-Tools) của Google, 1 thư viện có cấu trúc rất tốt & dễ sử dụng.

OR-Tools library is comprehensive. It offers best interface I have ever used to access multiple linear & integer solvers (MPSolver). It also has special-purpose code for network flow problems as well as a very effective constraint programming library. In this text, display only a very small fraction of this cornucopia of optimization tools.

– Thư viện OR-Tools rất toàn diện. Nó cung cấp giao diện tốt nhất mà tôi từng sử dụng để truy cập nhiều bộ giải tuyến tính & số nguyên (MPSolver). Nó cũng có mã chuyên dụng cho các bài toán luồng mạng cũng như 1 thư viện lập trình ràng buộc rất hiệu quả. Trong bài viết này, tôi chỉ trình bày 1 phần rất nhỏ trong kho tàng công cụ tối ưu hóa này.

1 of many advantages of using a general purpose language like Python: we can do modeling part as well as insertion of models into a larger application, maybe a web or a phone app. Can also easily present solutions in a clear format. We have all power of a complete language at our disposal. True, specialized modeling languages sometimes allow more concise model expression. But, in my experience, they all, at 1 point or another, hit a wall, forcing modeler to write kludgy glue to connect a model to rest of application. Moreover, writing OR-Tools models in Python can be such a joy. [Writing in Common Lisp would be even better. Alas, there is no Lisp binding for OR-Tools yet.] Whole coexistence model is shown at Listing 1.1: Amphibian Coexistence Model.

– 1 trong nhiều lợi thế của việc sử dụng 1 ngôn ngữ đa năng như Python: chúng ta có thể thực hiện cả phần mô hình hóa cũng như chèn mô hình vào 1 ứng dụng lớn hơn, có thể là 1 trang web hoặc 1 ứng dụng điện thoại. Cũng có thể dễ dàng trình bày các giải pháp theo 1 định dạng rõ ràng. Chúng ta có tất cả sức mạnh của 1 ngôn ngữ hoàn chỉnh theo ý mình. Các ngôn ngữ mô hình hóa chuyên biệt thực sự đôi khi cho phép biểu diễn mô hình ngắn gọn hơn. Nhưng, theo kinh nghiệm của tôi, tất cả chúng, tại 1 thời điểm nào đó, đều gặp phải 1 bức tường, buộc người tạo mô hình phải viết keo dán kludgy để kết nối 1 mô hình với phần còn lại của ứng dụng. Hơn nữa, việc viết các mô hình OR-Tools bằng Python có thể rất thú vị. [Viết bằng Common Lisp thậm chí còn tốt hơn. Than ôi, vẫn chưa có ràng buộc Lisp nào cho OR-Tools.] Toàn bộ mô hình cùng tồn tại được hiển thị tại Liệt kê 1.1: Mô hình cùng tồn tại của lưỡng cư.

Deconstruct code. Line 1 loads Python wrapper of linear programming subset of OR-Tools. Every model we write will start this way. Line 4 names & creates a linear programming solver (hereafter named s) using Google's own ˜https://developers.google.com/optimization/lp/glop GLOP. OR-Tools library has interfaces to a number of solvers. Switching to a different solver, say GNU's www.gnu.org/software/glpk/ GLPK or Coin-or ˜https://projects.coin-or.org/Clp CLP is a simple matter or modifying this line.

– Phân tích mã. Dòng 1 tải trình bao bọc Python của tập con lập trình tuyến tính của OR-Tools. Mọi mô hình chúng ta viết sẽ bắt đầu theo cách này. Dòng 4 đặt tên & tạo 1 trình giải lập trình tuyến tính (sau đây gọi là s) sử dụng https://developers.google.com/optimization/lp/glop GLOP của Google. Thư viện OR-Tools có giao diện với 1 số trình giải. Việc chuyển sang 1 trình giải khác, chẳng hạn như www.gnu.org/software/glpk/ của GNU, GLPK hoặc Coin-or https://projects.coin-or.org/Clp CLP chỉ là 1 vấn đề đơn giản hoặc sửa đổi dòng này.

On line 5, create a 1D array `x` of 3 decision variables that can take on values between 0 & 1000. Lower bound is a physical constraint since we cannot have a negative number of amphibians. Upper bound is part of problem statement as biologist will not put > 1000 of each species in test tube. Possible to state ranges as any contiguous subsets of $(-\infty, \infty)$, but, as a general rule of thumb, restricting range as much as possible during variable declaration tends to help solvers run efficiently. 3rd parameter of call to `NumVar` is used as name to print if & when this variable is displayed, e.g., in debugging a model. Will have little use for this feature as prefer to write bug-free models.

– Ở dòng 5, tạo 1 mảng 1 chiều `x` gồm 3 biến quyết định có thể nhận các giá trị từ 0 & 1000. Giới hạn dưới là 1 ràng buộc vật lý vì chúng ta không thể có số lượng lưỡng cư âm. Giới hạn trên là 1 phần của phát biểu bài toán vì nhà sinh vật học sẽ không đưa > 1000 của mỗi loài vào ống nghiệm. Có thể nêu các phạm vi là bất kỳ tập hợp con liền kề nào của $(-\infty, \infty)$, nhưng, theo nguyên tắc chung, việc hạn chế phạm vi càng nhiều càng tốt trong quá trình khai báo biến có xu hướng giúp trình giải chạy hiệu quả. Tham số thứ 3 của lệnh gọi tới `NumVar` được sử dụng làm tên để in nếu & khi biến này được hiển thị, ví dụ: khi gỡ lỗi mô hình. Sẽ ít sử dụng tính năng này vì thích viết các mô hình không có lỗi.

Constraints on lines 7–9 are direct translations of mathematical expressions (1.1)–(1.3). Order of terms is irrelevant. In contrast to some restrictive modeling languages, we could have written line 7 as `1500 >= x[0] + x[2] + x[1]` or `x[0] + x[1] + x[2] - 1500 <= 0` or any other equivalent algebraic expression.

– Các ràng buộc trên dòng 7-9 là bản dịch trực tiếp của các biểu thức toán học (1.1)–(1.3). Thứ tự các thuật ngữ không liên quan. Trái ngược với 1 số ngôn ngữ mô hình hóa hạn chế, chúng ta có thể viết dòng 7 là `1500 >= x[0] + x[2] + x[1]`

hoặc `x[0] + x[1] + x[2] - 1500 <= 0` hoặc bất kỳ biểu thức đại số tương đương nào khác.

At line 6, declare an auxiliary variable `pop`. Though there is no such distinction in modeling language, this is not a decision variable but rather a helpful device to model problem. Use this auxiliary on line 10 where we add an equation that does not constrain model in any way. It simply defines auxiliary variable `pop` to be sum of our decision variables. This allows us to express objective easily &, possibly, to help display solution.

– Ở dòng 6, hãy khai báo 1 biến phụ `pop`. Mặc dù không có sự phân biệt như vậy trong ngôn ngữ mô hình hóa, nhưng đây không phải là 1 biến quyết định mà là 1 công cụ hữu ích để mô hình hóa bài toán. Hãy sử dụng biến phụ này ở dòng 10, nơi chúng ta thêm 1 phương trình không ràng buộc mô hình theo bất kỳ cách nào. Nó chỉ đơn giản định nghĩa biến phụ `pop` là tổng của các biến quyết định. Điều này cho phép chúng ta dễ dàng thể hiện mục tiêu &, có thể, giúp hiển thị lời giải.

Objective function is on line 11, a translation of (1.4). Function choices are, unsurprisingly, either `s.Maximize` or `s.Minimize` with, for parameter, a linear expression in terms of variables declared previously. Used `s.Maximize(pop)`. Could have written `s.Maximize(x[0] + x[1] + x[2])`. Then call on solver at line 12 to do its job. This is where all computational work gets done, work that I will not describe. Interested reader can search for "simplex method" & "interior-point methods" to learn about fascinating theory [See, e.g., Alexander Schrijver, Theory of Linear & Integer Programming (Hoboken, NJ: Wiley, 1998).] behind solution methods of linear optimization models. To understand simplex method, one needs only high school algebra. To understand interior-point methods requires a somewhat more mathematical background.

– Hàm mục tiêu nằm ở dòng 11, bản dịch của (1.4). Không có gì ngạc nhiên khi các lựa chọn hàm là `s.Maximize` hoặc `s.Minimize` với tham số là 1 biểu thức tuyến tính theo các biến đã khai báo trước đó. Đã sử dụng `s.Maximize(pop)`. Có thể viết `s.Maximize(x[0] + x[1] + x[2])`. Sau đó, gọi trình giải ở dòng 12 để thực hiện công việc của nó. Đây là nơi thực hiện tất cả các công việc tính toán, công việc mà tôi sẽ không mô tả. Độc giả quan tâm có thể tìm kiếm "phương pháp đơn hình" & "phương pháp điểm trong" để tìm hiểu về lý thuyết hấp dẫn [Xem, ví dụ: Alexander Schrijver, Lý thuyết về Lập trình Tuyến tính & Số nguyên (Hoboken, NJ: Wiley, 1998).] đằng sau các phương pháp giải của các mô hình tối ưu hóa tuyến tính. Để hiểu phương pháp đơn hình, người ta chỉ cần kiến thức đại số trung học phổ thông. Để hiểu được các phương pháp điểm bên trong đòi hỏi phải có nền tảng toán học nhiều hơn 1 chút.

For some models, solvers may complete their work in a fraction of a second; for others, it may take hours. Moreover, not all solvers will have same runtime behavior. Model A may run faster than model B on solver X while it may be exactly reverse on solver Y. 1 more advantage of using OR-Tools library: can try out another solver by changing 1 line.

– Với 1 số mô hình, trình giải có thể hoàn thành công việc chỉ trong 1 phần giây; với 1 số mô hình khác, có thể mất hàng giờ. Hơn nữa, không phải tất cả trình giải đều có cùng hành vi thời gian chạy. Mô hình A có thể chạy nhanh hơn mô hình B trên trình giải X, nhưng có thể hoàn toàn ngược lại trên trình giải Y. 1 lợi thế nữa của việc sử dụng thư viện OR-Tools: có thể thử nghiệm 1 trình giải khác bằng cách thay đổi 1 dòng.

We should, if this code were meant for production & problem nontrivial, check return value to ensure: solver found an optimal solution. It may have aborted because of a model error, or because it ran out of time or memory, or for some other reason. But for this simple 1st example, we will forgo good engineering practice in name of simplicity of exposition.

– Nếu đoạn mã này được dùng cho mục đích sản xuất & bài toán không tầm thường, chúng ta nên kiểm tra giá trị trả về để đảm bảo: trình giải đã tìm thấy giải pháp tối ưu. Nó có thể bị dừng lại do lỗi mô hình, hoặc do hết thời gian hoặc bộ nhớ, hoặc vì lý do nào khác. Nhưng đối với ví dụ đơn giản đầu tiên này, chúng ta sẽ bỏ qua các phương pháp kỹ thuật tốt để đơn giản hóa việc trình bày.

Return, on line 13, both optimal objective function value held in variable `pop` & optimal values of decision variables (not all associated object attributes carried by those variables).

– Trả về, trên dòng 13, cả giá trị hàm mục tiêu tối ưu được lưu giữ trong biến `pop` & giá trị tối ưu của các biến quyết định (không phải tất cả các thuộc tính đối tượng liên quan được các biến đó mang theo).

On more complex models, we may post-process decision variables to return something simpler & more meaningful to caller. You will see a good example of this when solve shortest path problem in Chap. 4, Sect. 4.4. General approach I encourage: create models that can be used without any knowledge of internals of OR-Tools. Modeler is responsible for creation of model, but once model is created & validated, it should leave hands of its creator for those of domain expert who originally formulated problem. When diligent reader executes Listing 1.2: How to Execute Coexistence Model, will observe a result similar to Table 1.3: Solution to Coexistence Problem.

– Trên các mô hình phức tạp hơn, chúng ta có thể xử lý hậu kỳ các biến quyết định để trả về 1 kết quả đơn giản hơn & có ý nghĩa hơn đối với người gọi. Bạn sẽ thấy 1 ví dụ điển hình về điều này khi giải bài toán đường đi ngắn nhất trong Chương 4, Mục 4.4. Tôi khuyến khích cách tiếp cận chung: tạo các mô hình có thể sử dụng mà không cần bất kỳ kiến thức nào về OR-Tools. Người tạo mô hình chịu trách nhiệm tạo mô hình, nhưng sau khi mô hình được tạo & xác thực, nó nên được chuyển giao cho những chuyên gia trong lĩnh vực đã xây dựng bài toán ban đầu. Khi người đọc cẩn thận thực hiện Liệt kê 1.2: Cách thực thi Mô hình Đồng tồn tại, sẽ thấy kết quả tương tự như Bảng 1.3: Giải pháp cho Bài toán Đồng tồn tại.

Notice can look at solution of Table 1.3 & see it does indeed satisfy constraints. Notice: 1st 2 inequalities are satisfied with equality. In jargon of optimization, such inequalities are *tight* or *active*. Last one is said to be *slack* or *inactive*. In a certain sense, we could delete it from problem & nothing would change. (Reader can try this & other modifications. Code is available in additional material under name `coexistence.py`).

– Lưu ý: Có thể xem xét lời giải của Bảng 1.3 & thấy rằng nó thực sự thỏa mãn các ràng buộc. Lưu ý: 2 bất đẳng thức đầu tiên được thỏa mãn bằng đẳng thức. Theo thuật ngữ tối ưu hóa, các bất đẳng thức này được gọi là *tight* hoặc *active*. Bất đẳng thức cuối cùng được gọi là *slack* hoặc *inactive*. Theo 1 nghĩa nào đó, chúng ta có thể xóa nó khỏi bài toán & không có gì thay đổi. (Bạn đọc có thể thử cách này & các sửa đổi khác. Mã có sẵn trong tài liệu bổ sung dưới tên `coexistence.py`).

In summary, steps to construct & run a model are following & are shown in Fig. 1.1: Steps to construct & run a model.:

1. Formulate question precisely.
2. Define decision variables by identifying what is required to answer question.
3. Possibly define auxiliary variables to help simplify statements of constraints or of objective function. They can also help in analysis & presentation of solution.
4. Translate each constraint into an algebraic equality or inequality involving directly decision variables or indirectly through auxiliary variables.
5. Construct objective function as some quantity that should be minimized or maximized.
6. Run model using an appropriate solver.
7. Display solution in an appropriate manner.
8. Validate results. Does solution correctly satisfy constraints? Is solution meaningful & implementable? If so, declare that you are done; if not, consider necessary modifications to model.

Rest of this book will construct models of increasingly complexity, illustrating & expanding points above.

– Tóm lại, các bước để xây dựng & chạy mô hình như sau & được thể hiện trong Hình 1.1: Các bước để xây dựng & chạy mô hình.:

1. Đặt câu hỏi 1 cách chính xác.
2. Xác định các biến quyết định bằng cách xác định những gì cần thiết để trả lời câu hỏi.
3. Có thể xác định các biến phụ trợ để giúp đơn giản hóa các phát biểu về ràng buộc hoặc hàm mục tiêu. Chúng cũng có thể hỗ trợ phân tích & trình bày lời giải.
4. Chuyển mỗi ràng buộc thành 1 đẳng thức hoặc bất đẳng thức đại số liên quan trực tiếp đến các biến quyết định hoặc gián tiếp thông qua các biến phụ trợ.
5. Xây dựng hàm mục tiêu dưới dạng 1 đại lượng nào đó cần được tối thiểu hóa hoặc tối đa hóa.
6. Chạy mô hình bằng 1 trình giải thích hợp.
7. Hiển thị lời giải theo cách thích hợp.
8. Xác thực kết quả. Lời giải có đáp ứng đúng các ràng buộc không? Lời giải có ý nghĩa & khả thi không? Nếu có, hãy tuyên bố bạn đã hoàn thành; nếu không, hãy xem xét các sửa đổi cần thiết cho mô hình.

Phần còn lại của cuốn sách này sẽ xây dựng các mô hình có độ phức tạp ngày càng tăng, minh họa & mở rộng các điểm trên.

- 2. Linear Continuous Models. At dawn of optimization (1950s), state-of-art was defined by linear optimization models & simplex method, only reasonably efficient algorithm known at time to solve such models. When I started studying this subject, one repeatedly heard from multiple sources: $> 70\%$ of CPU cycles in world were devoted to running various simplex codes. Surely an exaggeration, but it is indicative of power of linear models. World is not linear, but sometimes a linear approximation is good enough.

  – Vào buổi bình minh của tối ưu hóa (những năm 1950), công nghệ tiên tiến nhất được định nghĩa bằng các mô hình tối ưu tuyến tính & phương pháp simplex, thuật toán duy nhất hiệu quả tương đối vào thời điểm đó để giải các mô hình như vậy. Khi tôi bắt đầu nghiên cứu chủ đề này, tôi đã nghe từ nhiều nguồn: $> 70\%$ chu kỳ CPU trên thế giới được dành cho việc chạy các mã simplex khác nhau. Chắc chắn là phóng đại, nhưng điều này cho thấy sức mạnh của các mô hình tuyến tính. Thế giới không tuyến tính, nhưng đôi khi 1 phép xấp xỉ tuyến tính cũng đủ tốt.

  More precisely, discuss here *linear continuous* models (although usage is to call these models LPs for linear programs, implying continuity properties). Linear continuous models are simplest to write down & simplest to solve. They have been workhorse of optimizers since George Dantzig invented simplex method to solve them. What characterizes them are 3 elements:

  1. All variables are continuous.
  2. All constraints are linear.
  3. Objective function is linear.

  In detail, decision variables, say $x_0, \ldots, x_n$ can take on integral & fractional values. This is appropriate when solution is measuring amounts (e.g., pounds of flour or tons of concrete). Not appropriate when solution is counting objects (as in people or politicians), unless one is looking only for an approximation.

  – Chính xác hơn, hãy thảo luận ở đây về các mô hình *tuyến tính liên tục* (mặc dù cách sử dụng là gọi các mô hình này là LP cho các chương trình tuyến tính, ngụ ý các tính chất liên tục). Các mô hình tuyến tính liên tục dễ viết nhất & dễ giải nhất. Chúng đã là công cụ đắc lực của các nhà tối ưu hóa kể từ khi George Dantzig phát minh ra phương pháp đơn hình để giải chúng. Đặc điểm của chúng là 3 yếu tố:

  1. Tất cả các biến đều liên tục.
  2. Tất cả các ràng buộc đều tuyến tính.
  3. Hàm mục tiêu là tuyến tính.

Cụ thể hơn, các biến quyết định, chẳng hạn như $x_0, \ldots, x_n$ có thể nhận các giá trị tích phân & phân số. Điều này phù hợp khi giải pháp là đo lường số lượng (ví dụ: pound bột mì hoặc tấn bê tông). Không phù hợp khi giải pháp là đếm các đối tượng (như con người hoặc chính trị gia), trừ khi người ta chỉ tìm kiếm 1 giá trị gần đúng.

Objective function is (or can be) parametrized by constant array c & expressed as $\sum_{i=1}^{n} c_i x_i$. This limitation precludes objective functions with terms of form $x_1^2, x_4^4, \sin x, e^{x_3}, x_1 x_2, |x|$ among infinitely many others, although see later how to handle some of these nonlinearities by model transformation.

– Hàm mục tiêu được (hoặc có thể) tham số hóa bởi mảng hằng số c & được biểu diễn dưới dạng $\sum_{i=1}^{n} c_i x_i$. Giới hạn này ngăn cản các hàm mục tiêu có dạng $x_1^2, x_4^4, \sin x, e^{x_3}, x_1 x_2, |x|$ cùng vô số dạng khác, mặc dù chúng ta sẽ xem cách xử lý 1 số tính phi tuyến tính này bằng phép biến đổi mô hình ở phần sau.

Finally, constraints are parameterized by matrix a_{ij}, array b, & can be stated as a set of relations, for $i \in [m]$,

$$\sum_{j=1}^{n} a_{ij} x_{ij} \geq b_i, \text{ or } \leq b_i, \text{ or } = b_i$$

or some equivalent algebraic form. In this chap, consider problems where natural formulation is such a linear continuous model.

– Cuối cùng, các ràng buộc được tham số hóa bởi ma trận a_{ij}, mảng b, & có thể được biểu diễn dưới dạng 1 tập hợp các quan hệ, với $i \in [m]$,

$$\sum_{j=1}^{n} a_{ij} x_{ij} \geq b_i, \text{ hoặc } \leq b_i, \text{ hoặc } = b_i$$

hoặc 1 dạng đại số tương đương nào đó. Trong chương này, hãy xem xét các bài toán mà công thức tự nhiên là 1 mô hình tuyến tính liên tục như vậy.

◦ 2.1. Mixing. Canonical linear programming example is *diet problem*, 1 of 1st optimization problems to be studied in 1930s & 1940s [I add this temporal precision on odd chance that this text is still being read long after my body has maximized its entropy.] Likely apocryphal origin of problem is US military's desire to meet nutritional requirements of field GIs while minimizing cost of food. 1 of early researchers to study this problem was *George Stigler*. He made an educated guess of optimal solution to linear program using a heuristic method. In fall of 1947, Jack Laderman of Mathematical Tables Project of National Bureau of Standards (NBS, today NIST) undertook solving Stigler's model with new simplex method. Linear model consisted of 9 equations in 77 unknowns, a huge problem for time. Some models in this book are orders of magnitude larger & will be solved in a minuscule fraction of time it took NBS people to solve diet problem in 1947. Increase in efficiency is partly due to hardware, but mostly due to software.

– Ví dụ về lập trình tuyến tính chuẩn là *diet problem*, 1 trong những bài toán tối ưu hóa đầu tiên được nghiên cứu vào những năm 1930 & 1940 [Tôi thêm độ chính xác về mặt thời gian này vào cơ hội kỳ lạ là văn bản này vẫn được đọc rất lâu sau khi cơ thể tôi đã đạt đến mức entropy tối đa.] Nguồn gốc không rõ ràng của vấn đề có thể là mong muốn của quân đội Hoa Kỳ nhằm đáp ứng nhu cầu dinh dưỡng của lính Mỹ tại chiến trường trong khi giảm thiểu chi phí thực phẩm. 1 trong những nhà nghiên cứu đầu tiên nghiên cứu vấn đề này là *George Stigler*. Ông đã đưa ra 1 phỏng đoán có căn cứ về giải pháp tối ưu cho lập trình tuyến tính bằng phương pháp heuristic. Vào mùa thu năm 1947, Jack Laderman của Dự án Bảng toán học thuộc Cục Tiêu chuẩn Quốc gia (NBS, ngày nay là NIST) đã đảm nhận việc giải mô hình của Stigler bằng phương pháp đơn hình mới. Mô hình tuyến tính bao gồm 9 phương trình trong 77 ẩn số, 1 bài toán lớn về thời gian. 1 số mô hình trong cuốn sách này có quy mô lớn hơn rất nhiều & sẽ được giải quyết trong 1 khoảng thời gian cực ngắn so với thời gian mà những người NBS giải quyết vấn đề chế độ ăn uống vào năm 1947. Hiệu quả tăng lên 1 phần là do phần cứng, nhưng chủ yếu là do phần mềm.

A generic version of problem is: Given a list of food with some nutritional content, each with a cost, find combination of food that will minimize cost & yet provide all necessary nutrients. Here is 1 simple version of this problem. Foods are $F_0, F_1, \ldots$ (Imagine them to be pizza, ramen noodles, cupcakes, chips, etc. or, if you are of a more health-conscious bent, tofu, green peas, quinoa, beets, etc.) Nutrients will be represented by $N_0, N_1, \ldots$ (Imagine them to be calories, protein, calcium, vitamin A, etc.). Each has a cost per serving. In addition, to avoid eating 1 food all week long, restrict number of servings per week.

– 1 phiên bản chung của bài toán này là: Cho 1 danh sách thực phẩm có chứa 1 số chất dinh dưỡng, mỗi loại có 1 giá trị, hãy tìm sự kết hợp thực phẩm sao cho chi phí thấp nhất & vẫn cung cấp đầy đủ các chất dinh dưỡng cần thiết. Đây là 1 phiên bản đơn giản của bài toán này. Thực phẩm là $F_0, F_1, \ldots$ (Hãy tưởng tượng chúng là pizza, mì ramen, bánh nướng nhỏ, khoai tây chiên, v.v. hoặc, nếu bạn quan tâm đến sức khỏe hơn, đậu phụ, đậu xanh, hạt diêm mạch, củ cải đường, v.v.). Chất dinh dưỡng sẽ được biểu thị bằng $N_0, N_1, \ldots$ (Hãy tưởng tượng chúng là calo, protein, canxi, vitamin A, v.v.). Mỗi loại có giá trị trên mỗi khẩu phần. Ngoài ra, để tránh ăn 1 loại thực phẩm trong suốt cả tuần, hãy hạn chế số lượng khẩu phần ăn mỗi tuần.

A randomly generated instance is given in Table 2.1: Example of Data & Solution for Diet Problem. Each row represents a food, with nutritional content per serving followed by acceptable range of servings of food & its cost per serving. Ignore last row & column for now. Return to them after model is constructed & solved. 2 rows before last represent allowed range of each nutrient.

– 1 ví dụ được tạo ngẫu nhiên được đưa ra trong Bảng 2.1: Ví dụ về Dữ liệu & Giải pháp cho Bài toán Chế độ ăn uống. Mỗi hàng đại diện cho 1 loại thực phẩm, với hàm lượng dinh dưỡng trên mỗi khẩu phần, theo sau là phạm vi khẩu phần ăn được chấp nhận & chi phí cho mỗi khẩu phần. Bỏ qua hàng cuối cùng & cột. Quay lại chúng sau khi mô hình được xây dựng & giải quyết. 2 hàng trước hàng cuối cùng đại diện cho phạm vi cho phép của mỗi chất dinh dưỡng.

∗ 2.1.1. Constructing a Model. What would a solution be but a list of servings of each food? Therefore, decision variables must be one per food, representing number of servings. Name these variables $f_0, \ldots, f_n$. Assume: acceptable to have fractional answers (i.e., 1 half serving is acceptable).
– Xây dựng Mô hình. Giải pháp sẽ là gì nếu không phải là danh sách khẩu phần ăn của mỗi loại thực phẩm? Do đó, các biến quyết định phải là 1 cho mỗi loại thực phẩm, biểu diễn số khẩu phần ăn. Đặt tên cho các biến này là $f_0, \ldots, f_n$. Giả sử: có thể chấp nhận được các đáp án phân số (tức là, 1 nửa khẩu phần ăn là chấp nhận được).
Objective: minimize cost. Have 1 cost per food $c_0, \ldots, c_n$. These are not variables, they are data. Therefore, what we want is to minimize sum of all products $c_i f_i$. This leads to objective function

$$\min \sum_i c_i f_i.$$

Tackle constraints. Have 2 sets: one indicating range of acceptable servings of each food (assume: minimum of food i is l_i & maximum is u_i) & one indicating required nutrients range (minimum of nutrient j is a_j & maximum is b_j). Simpler constraint is related to food. Since our decision variables indicate number of servings of each food, need only to box each serving count

$$l_i \leq f_i \leq u_i.$$

Constraint on nutrients is a bit more involved. Consider nutrient j. How much of it will be included in diet? Each food i may have some of it, as indicated in Table 2.1. Call this amount N_{ji} (corresponding to entry at row of food i & column of nutrient j). To get total of this nutrient, therefore need to sum over all foods product of food serving & nutrient content. For each nutrient j,

$$a_j \leq \sum_i N_{ji} f_i \leq b_j.$$

We are done with theory. Translate this into an executable model general enough to solve all problems of this type Listing 2.1: Model for Minimal Cost Diet. Assume: data is given in a 2D array called `N`. It has structure of Table 2.1 without last column & row. Each row represents a food, except last 2 rows represent minimum & maximum requirement of each nutrient, represented by columns, with last 3 representing minimum, maximum, & cost each food serving.
– Mục tiêu: giảm thiểu chi phí. Có 1 chi phí cho mỗi loại thực phẩm $c_0, \ldots, c_n$. Đây không phải là biến, mà là dữ liệu. Do đó, điều chúng ta muốn là giảm thiểu tổng của tất cả các tích $c_i f_i$. Điều này dẫn đến hàm mục tiêu

$$\min \sum_i c_i f_i.$$

Xử lý các ràng buộc. Có 2 tập hợp: 1 tập hợp biểu thị phạm vi khẩu phần ăn được chấp nhận của mỗi loại thực phẩm (giả sử: giá trị nhỏ nhất của thực phẩm i là l_i & giá trị lớn nhất là u_i) & 1 tập hợp biểu thị phạm vi dinh dưỡng cần thiết (giá trị nhỏ nhất của dinh dưỡng j là a_j & giá trị lớn nhất là b_j). Ràng buộc đơn giản hơn liên quan đến thực phẩm. Vì các biến quyết định của chúng ta biểu thị số khẩu phần ăn của mỗi loại thực phẩm, chỉ cần đóng khung mỗi số khẩu phần

$$l_i \leq f_i \leq u_i.$$

Ràng buộc về chất dinh dưỡng phức tạp hơn 1 chút. Hãy xem xét chất dinh dưỡng j. Bao nhiêu chất dinh dưỡng sẽ được đưa vào chế độ ăn? Mỗi loại thực phẩm i có thể chứa 1 phần chất dinh dưỡng này, như được chỉ ra trong Bảng 2.1. Gọi lượng này là N_{ji} (tương ứng với mục nhập ở hàng thực phẩm i & cột chất dinh dưỡng j). Để có tổng chất dinh dưỡng này, cần cộng tất cả các loại thực phẩm với tích của khẩu phần ăn & hàm lượng chất dinh dưỡng. Đối với mỗi chất dinh dưỡng j,

$$a_j \leq \sum_i N_{ji} f_i \leq b_j.$$

Vậy là xong phần lý thuyết. Hãy chuyển đổi điều này thành 1 mô hình thực thi đủ tổng quát để giải quyết tất cả các bài toán thuộc loại này Liệt kê 2.1: Mô hình cho Chế độ ăn uống Chi phí Tối thiểu. Giả sử: dữ liệu được đưa ra trong 1 mảng 2 chiều có tên là `N`. Nó có cấu trúc như Bảng 2.1 nhưng không có cột cuối cùng & hàng. Mỗi hàng đại diện cho 1 loại thực phẩm, ngoại trừ 2 hàng cuối đại diện cho nhu cầu tối thiểu & tối đa của từng chất dinh dưỡng, được biểu thị bằng các cột, trong đó 3 hàng cuối đại diện cho nhu cầu tối thiểu, tối đa & chi phí của mỗi khẩu phần thực phẩm.
Model uses `newSolver` function to simplify expression of code [Mostly to make code fit a page, but also to hide some of verbosity of OR-Tools library. Authors chose, rightly in my opinion, meaningful but rather long names for their functions.] as reader can see at Listing 2.2: Utility Function to Create an Appropriate Solver Instance. These, & other simplifications, can be found in `my_or_tools.py`.
– Mô hình sử dụng hàm `newSolver` để đơn giản hóa biểu thức mã [Chủ yếu là để mã vừa với 1 trang, nhưng cũng để ẩn bớt sự rườm rà của thư viện OR-Tools. Theo tôi, tác giả đã chọn đúng tên hàm có ý nghĩa nhưng khá dài.] như người đọc có thể thấy tại Liệt kê 2.2: Hàm Tiện ích để Tạo 1 Thể hiện Trình giải Thích hợp. Những hàm này, cùng với các cách đơn giản hóa khác, có thể được tìm thấy trong `my_or_tools.py`.
To help expression of model, lines 3–4 give meaningful names to row & column indices that we will use. In line 5, define decision variables, 1 per food, each taking values in range $[l_i, u_i]$ as in (2.1). It would be correct to give a rang of $[0, \infty)$ & then add constraints to enforce bounds. Solver would still find same solution, but it is simpler & good practice to limit as much as possible range of decision variables. In complex models, it often dramatically improves solution time.

– Để hỗ trợ biểu diễn mô hình, các dòng 3-4 đặt tên có ý nghĩa cho các chỉ số hàng & cột mà chúng ta sẽ sử dụng. Ở dòng 5, hãy định nghĩa các biến quyết định, mỗi biến 1 cho mỗi loại thực phẩm, mỗi biến lấy giá trị trong khoảng $[l_i, u_i]$ như trong (2.1). Sẽ đúng khi đưa ra 1 khoảng $[0, \infty)$ & sau đó thêm các ràng buộc để áp đặt các giới hạn. Trình giải vẫn sẽ tìm ra cùng 1 giải pháp, nhưng việc hạn chế càng nhiều khoảng biến quyết định càng tốt sẽ đơn giản hơn & là 1 thực hành tốt. Trong các mô hình phức tạp, điều này thường cải thiện đáng kể thời gian giải.

2-line loop starting on line 6 establishes range on each nutrient as in (2.1.1). Line 9 & following create objective function, solve problem, & return 3 numbers: status of solver (it should be 0), optimal value, & optimal solution. Dual role of functions `SolVal, ObjVal` (see in Listing 2.3: Utility Functions to Extract Values from OR-Tools Objects) is to simplify results returned to caller & code to read.

– Vòng lặp 2 dòng bắt đầu từ dòng 6 thiết lập phạm vi của mỗi chất dinh dưỡng như trong (2.1.1). Dòng 9 & tiếp theo tạo hàm mục tiêu, giải bài toán, & trả về 3 số: trạng thái của trình giải (phải bằng 0), giá trị tối ưu, & giải pháp tối ưu. Vai trò kép của các hàm `SolVal, ObjVal` (xem trong Liệt kê 2.3: Các hàm tiện ích để trích xuất giá trị từ các đối tượng OR-Tools) là đơn giản hóa kết quả trả về cho trình gọi & mã để đọc.

Results from executing this model are shown in last row & column of Table 2.1. Column indicates number of servings of each food & row indicates amount of each nutrient that will be in diet. Reader should notice that many of food items & nutrient counts are at their minimum required values. This is expected of such a model since we are trying to minimize a linear cost function; optimal solution should push towards boundary of constraints as much as possible.

– Kết quả thực hiện mô hình này được hiển thị ở hàng cuối cùng & cột của Bảng 2.1. Cột biểu thị số khẩu phần của mỗi loại thực phẩm & hàng biểu thị lượng chất dinh dưỡng sẽ có trong chế độ ăn. Người đọc nên lưu ý rằng nhiều loại thực phẩm & lượng chất dinh dưỡng đều đạt giá trị tối thiểu cần thiết. Điều này là bình thường đối với 1 mô hình như vậy vì chúng ta đang cố gắng tối thiểu hóa hàm chi phí tuyến tính; giải pháp tối ưu nên tiến gần đến ranh giới ràng buộc càng nhiều càng tốt.

Reader can experiment with this model. It is included in additional material as `diet_problem.py`, along with a generator of random diet problems & a routine to display solution in a table format similar to Table 2.1.

– Người đọc có thể thử nghiệm mô hình này. Nó được bao gồm trong tài liệu bổ sung dưới dạng `diet_problem.py`, cùng với trình tạo các bài toán chế độ ăn uống ngẫu nhiên & 1 chương trình để hiển thị giải pháp dưới dạng bảng tương tự như Bảng 2.1.

∗ 2.1.2. Variations. There are a number of simple variations of this problem.

1. Instead of minimizing cost, we could be given a profit to maximize. We could also not have either minima or maxima in either foods or nutrients.
   – Thay vì giảm thiểu chi phí, chúng ta có thể được trao lợi nhuận để tối đa hóa. Chúng ta cũng không thể có mức tối thiểu hoặc tối đa trong cả thực phẩm & chất dinh dưỡng.
2. It becomes more complex, & consequently interesting, when we have, in addition, requirements of form "If food 2 is used, then we must have at least as much food 3 in diet" or "Nutrient 3 must be included in at least twice amount as nutrient 4."
   – Vấn đề trở nên phức tạp hơn, & do đó thú vị hơn, khi chúng ta có thêm các yêu cầu theo dạng "Nếu sử dụng thực phẩm 2, thì chúng ta phải có ít nhất lượng thực phẩm 3 trong chế độ ăn" hoặc "Chất dinh dưỡng 3 phải được bổ sung với lượng ít nhất gấp đôi chất dinh dưỡng 4."
   Consider some of these in detail. 1st, try "If food 2 is used, then food 3 must also be included in at least as many servings." Following inequality ensures required result: $f_3 \geq f_2$. Notice food 3 could still be included when food 2 is not, but that does not violated requirement. & if food 2 is included, then we will have at least as many servings of food 3. It should be clear: requirement could have been stated in reverse as "No more food 2 than food 3." Constraint is same.
   – Hãy xem xét chi tiết 1 số điều sau. Trước tiên, hãy thử "Nếu sử dụng thức ăn 2, thì thức ăn 3 cũng phải được đưa vào ít nhất bằng số khẩu phần ăn đó." Bất đẳng thức sau đảm bảo kết quả cần thiết: $f_3 \geq f_2$. Lưu ý rằng thức ăn 3 vẫn có thể được đưa vào khi thức ăn 2 không được đưa vào, nhưng điều đó không vi phạm yêu cầu. & nếu thức ăn 2 được đưa vào, thì chúng ta sẽ có ít nhất bằng số khẩu phần ăn 3. Rõ ràng: yêu cầu có thể được phát biểu ngược lại là "Không có thức ăn 2 nào nhiều hơn thức ăn 3." Ràng buộc vẫn giữ nguyên.
3. A requirement on nutrients, "Nutrient 3 must be included in at least twice amount as nutrient 4," has a similar flavor but note: amount of any given nutrient is spread among all food items. It may be fruitful to introduce auxiliary variables that will tally nutrients, say n_j. Then add to model 1 equality per nutrient $n_j \leq \sum_i N_{ij} f_i$. Note these equations do not constrain problem; their insertion is simply a helpful device to implement requirement. We can now easily relate nutrient content according to new requirement as $n_3 \geq 2n_4$. This we could have stated, had we not defined variables n_i as
$$\sum_i N_{i3} f_i \geq 2 \sum_i N_{i4} f_i.$$
   Defining auxiliary variables n_j seems clearer. Moreover, displaying total of each nutrient at end might help with analysis or presentation of solution.
   – 1 yêu cầu về chất dinh dưỡng, "Chất dinh dưỡng 3 phải được bổ sung với lượng ít nhất gấp đôi chất dinh dưỡng 4", có nội dung tương tự nhưng lưu ý: lượng của bất kỳ chất dinh dưỡng nào được phân bổ đều cho tất cả các loại thực phẩm. Có thể hữu ích khi đưa vào các biến phụ trợ để tính tổng các chất dinh dưỡng, chẳng hạn như n_j. Sau đó, thêm vào mô hình 1 sự bằng nhau của mỗi chất dinh dưỡng $n_j \leq \sum_i N_{ij} f_i$. Lưu ý rằng các phương trình này không ràng buộc bài toán; việc chèn chúng chỉ đơn giản là 1 công cụ hữu ích để thực hiện yêu cầu. Giờ đây, chúng ta có thể dễ

dàng liên hệ hàm lượng chất dinh dưỡng theo yêu cầu mới là $n_3 \geq 2n_4$. Chúng ta có thể đã nói điều này nếu chúng ta không định nghĩa các biến n_i là

$$\sum_i N_{i3} f_i \geq 2 \sum_i N_{i4} f_i.$$

Việc định nghĩa các biến phụ trợ n_j có vẻ rõ ràng hơn. Hơn nữa, việc hiển thị tổng lượng chất dinh dưỡng ở cuối có thể giúp phân tích hoặc trình bày giải pháp.

4. A similar requirement may occur to reader, namely "If food (nutrient) 3 is used then food (nutrient) 4 must not be (& vice versa)." This may look like a simple variation to above but it is decidedly not simple. If fact, it forces modeler to use a different modeling technique. Will see how to implement such requirements in later chaps (see, e.g., Sect. 7.2 in Chap. 7). There are 2 valid approaches to modeling such requirements properly: integer programming & constraint programming. Reader is encouraged to spend some time trying to model such constraints to develop intuition into difficulties. Key, & reason that this is a beast of an entirely different ilk: change is not uniquely quantitative (as much as, or twice amount of) but is additionally qualitative: transition between having an element & not having that element.
– Người đọc có thể nảy sinh 1 yêu cầu tương tự, cụ thể là "Nếu sử dụng thức ăn (chất dinh dưỡng) 3 thì không được sử dụng thức ăn (chất dinh dưỡng) 4 (& ngược lại)." Điều này có vẻ như là 1 biến thể đơn giản của yêu cầu trên nhưng thực tế không hề đơn giản. Trên thực tế, nó buộc người lập mô hình phải sử dụng 1 kỹ thuật lập mô hình khác. Sẽ xem cách triển khai các yêu cầu như vậy trong các chương sau (xem ví dụ: Mục 7.2 trong Chương 7). Có 2 cách tiếp cận hợp lệ để lập mô hình các yêu cầu như vậy 1 cách chính xác: lập trình số nguyên & lập trình ràng buộc. Người đọc được khuyến khích dành thời gian thử lập mô hình các ràng buộc như vậy để phát triển trực giác thành các khó khăn. Điểm mấu chốt & lý do tại sao đây là 1 con quái vật thuộc loại hoàn toàn khác: sự thay đổi không chỉ mang tính định lượng (nhiều như, hoặc gấp đôi số lượng) mà còn mang tính định tính: quá trình chuyển đổi giữa việc có 1 phần tử & không có phần tử đó.

∗ 2.1.3. Structure of Problems Under Consideration. Problems with structure of diet problem are generally known as *product mix* problems. They can be presented in various ways but if they can be fitted into abstract Table 2.2: Abstract Structure of Product Mix Problems they can all be handled in manner described in this sect. Of course, it may be that some of columns or rows are missing (no cost, or no price, or no maximum demand, etc.) that only simplifies model.
– Cấu trúc của các bài toán đang được xem xét. Các bài toán về cấu trúc của chế độ ăn uống thường được gọi là các bài toán hỗn hợp sản phẩm. Chúng có thể được trình bày theo nhiều cách khác nhau, nhưng nếu chúng có thể được đưa vào Bảng 2.2: Cấu trúc trừu tượng của các bài toán hỗn hợp sản phẩm, tất cả đều có thể được xử lý theo cách được mô tả trong phần này. Tất nhiên, có thể có 1 số cột hoặc hàng bị thiếu (không có chi phí, không có giá, hoặc không có nhu cầu tối đa, v.v.) mà chỉ đơn giản hóa mô hình.
Decision variable indicates amount of product needed & constraints indicate availabilities of raw material or, equivalently, capacities of processing units as well as demand bounds. Objectives are often profits to maximize or costs to minimize or, simply, quantity to produce.
– Biến quyết định biểu thị lượng sản phẩm cần thiết & các ràng buộc biểu thị khả năng cung cấp nguyên liệu thô hoặc tương đương, năng lực của các đơn vị chế biến cũng như giới hạn nhu cầu. Mục tiêu thường là lợi nhuận cần tối đa hóa hoặc chi phí cần giảm thiểu, hoặc đơn giản là số lượng cần sản xuất.
Here are a few instances to help reader recognize underlying structure. Reader is encouraged to marshal problems into format of Table 2.2 by inventing numbers.
– Dưới đây là 1 vài ví dụ giúp người đọc nhận ra cấu trúc cơ bản. Người đọc được khuyến khích sắp xếp các bài toán theo định dạng của Bảng 2.2 bằng cách tự nghĩ ra các con số.

1. A factory is producing cement of various types. Each product is composed of same elements, but in various quantities, & we have on hand a limited supply of each of these elements, each with a cost. To each final product is associated a profit. What is best mix of product to produce to maximize profit?
– 1 nhà máy đang sản xuất xi măng các loại. Mỗi sản phẩm được cấu thành từ cùng 1 thành phần, nhưng với số lượng khác nhau, & chúng tôi có sẵn 1 lượng cung hạn chế cho mỗi thành phần này, mỗi thành phần đều có chi phí riêng. Mỗi sản phẩm cuối cùng đều đi kèm với lợi nhuận. Vậy nên sản xuất loại sản phẩm nào để tối đa hóa lợi nhuận?
2. A Florida-based fruit company produces orange drinks, juices, & concentrates for various markets. Raw materials for all products are oranges, sugar, water, & time in various quantities, some positive & some negative (producing orange drinks requires water; producing concentrates generates water). Given certain availabilities, how much can company produce to maximize profit?
– 1 công ty trái cây có trụ sở tại Florida sản xuất nước cam, nước ép, & cô đặc cho nhiều thị trường khác nhau. Nguyên liệu thô cho tất cả các sản phẩm là cam, đường, nước, & thời gian với số lượng khác nhau, 1 số dương & âm (sản xuất nước cam cần nước; sản xuất cô đặc tạo ra nước). Với 1 số lượng nhất định, công ty có thể sản xuất bao nhiêu để tối đa hóa lợi nhuận?
3. A toy manufacturer produces a number of different toys. Each is composed of a number of basic materials &, in addition, requires special processing (assembling, painting, boxing). Processing is done on specialized machines & has a duration. Since manufacturer has limited supplies of materials & machines, which can only operate a certain number of hours per day, how many toys can be produced?
– 1 nhà sản xuất đồ chơi sản xuất 1 số loại đồ chơi khác nhau. Mỗi loại được làm từ 1 số vật liệu cơ bản & ngoài ra còn cần quá trình gia công đặc biệt (lắp ráp, sơn, đóng hộp). Quá trình gia công được thực hiện trên các máy móc chuyên dụng & có thời gian nhất định. Vì nhà sản xuất có nguồn cung cấp vật liệu & máy móc hạn chế, chỉ có thể hoạt động trong 1 số giờ nhất định mỗi ngày, vậy có thể sản xuất được bao nhiêu đồ chơi?

4. A fertilizer company named Bush, Rove, & Company (BR & Co.) has 2 products: a high phosphate blend & a low phosphate blend. They are produced by mixing different raw materials in various quantities.
   – 1 công ty phân bón tên là Bush, Rove, & Company (BR & Co.) có 2 sản phẩm: hỗn hợp phosphate cao & hỗn hợp phosphate thấp. Chúng được sản xuất bằng cách trộn các nguyên liệu thô khác nhau với số lượng khác nhau.
   Company can procure, from its own subsidiaries, at most some amount of each raw material per day at a fixed internal cost. This cost includes labor, power, depreciation, delivery, bribes, etc. In addition, mixing process incurs a certain cost per ton for each product.
   – Công ty có thể mua từ các công ty con của mình tối đa 1 lượng nguyên liệu thô nhất định mỗi ngày với chi phí nội bộ cố định. Chi phí này bao gồm nhân công, điện, khấu hao, vận chuyển, hối lộ, v.v. Ngoài ra, quá trình trộn cũng phát sinh 1 khoản chi phí nhất định cho mỗi tấn sản phẩm.
   Both products are sold to a wholesaler, Fox Inc., at a fixed price. Moreover, wholesaler has agreed to buy all production BR & Co. can produce. How much of each fertilizer should it produce?
   – Cả hai sản phẩm đều được bán cho 1 nhà bán buôn, Fox Inc., với mức giá cố định. Hơn nữa, nhà bán buôn đã đồng ý mua toàn bộ sản lượng mà BR & Co. có thể sản xuất. Vậy công ty nên sản xuất bao nhiêu phân bón cho mỗi loại?
5. Queequeg sells half-kilo bags of coffee in 3 blends, House, Special, & Gourmet, which sell at different prices per bag. Each blend is made up of Colombian, Cuban, & Kenyan coffee beans in various proportions. Queequeg has on hand some Colombian, Cuban, & Kenyan. How much of each blend should it bag to maximize revenues?
   – Queequeg bán các bao cà phê nửa ký với 3 loại pha trộn: House, Special, & Gourmet, với giá khác nhau cho mỗi bao. Mỗi loại pha trộn được làm từ hạt cà phê Colombia, Cuba & Kenya theo tỷ lệ khác nhau. Queequeg có sẵn 1 số loại cà phê Colombia, Cuba & Kenya. Vậy nên mua bao nhiêu mỗi loại để tối đa hóa doanh thu?

◦ 2.2. Blending. A 2nd type of problem that readily admits a linear model: *blending problem*. Classical example involves blending so-called raw or crude gasolines to achieve various refined products with specified octane value. E.g., assume: given crude gasoline $R_0, R_1, \ldots, R_n$, each with a certain octane rating, maximum availabilities in barrels, & a cost in dollars per barrels, as shown in Table 2.3: Example of Raw Gasolines for Blending Problem.

– Pha trộn. 1 loại bài toán thứ hai dễ dàng chấp nhận mô hình tuyến tính: bài toán pha trộn. Ví dụ điển hình liên quan đến việc pha trộn cái gọi là xăng thô hoặc xăng thô để tạo ra các sản phẩm tinh chế khác nhau với trị số octan xác định. Ví dụ, giả sử: cho xăng thô $R_0, R_1, \ldots, R_n$, mỗi loại có chỉ số octan nhất định, khả năng cung cấp tối đa tính theo thùng, & chi phí tính bằng đô la mỗi thùng, như được thể hiện trong Bảng 2.3: Ví dụ về Xăng thô cho Bài toán Pha trộn.

We are also given demands for multiple types of refined gasolines, (think Bronze, Silver, & Gold) with their own octane ratings. Demands are stated in minimum & maximum number of barrels along with their selling prices, as shown in Table 2.4: Example of Refined Gasolines for Blending Problem. Create 3 types by mixing appropriate raw gasolines together, assuming: octane rating of a mix is a linear function of volumes mixed. This is a crucial assumption: if mix half & half of octane ratings 80 & 90, get an octane rating of 85. If mix 40% of octane 80 & 60% of octane 90, get 86. This assumption is key to blending model.

– Chúng ta cũng được cung cấp nhu cầu về nhiều loại xăng tinh chế (ví dụ như Đồng, Bạc, & Vàng) với chỉ số octan riêng. Nhu cầu được thể hiện bằng số lượng thùng tối thiểu & tối đa cùng với giá bán của chúng, như thể hiện trong Bảng 2.4: Ví dụ về Xăng Tinh Chế cho Bài Toán Pha Chế. Tạo ra 3 loại bằng cách trộn các loại xăng thô thích hợp với nhau, giả sử: chỉ số octan của hỗn hợp là 1 hàm tuyến tính của thể tích đã trộn. Đây là 1 giả định quan trọng: nếu trộn 1 nửa & 1 nửa chỉ số octan 80 & 90, ta được chỉ số octan 85. Nếu trộn 40% octan 80 & 60% octan 90, ta được 86. Giả định này là chìa khóa cho mô hình pha chế.

Notice: there might be a number of ways to mix raw gasolines together to get required ratings. Our task: construct a model that will tell us exactly how to mix raw products to satisfy demands & maximize profits (understood as difference between total selling price of finished products & total cost of raw gasolines).

– Lưu ý: Có thể có nhiều cách pha trộn xăng thô để đạt được các tiêu chuẩn yêu cầu. Nhiệm vụ của chúng ta: xây dựng 1 mô hình cho biết chính xác cách pha trộn các sản phẩm thô để đáp ứng nhu cầu & tối đa hóa lợi nhuận (được hiểu là chênh lệch giữa tổng giá bán thành phẩm & tổng chi phí xăng thô).

∗ 2.2.1. Constructing a Model. What is question to answer? Ask this question a number of time with increasing precision. A 1st stab is "How much of each type of refined gas to produce?" This is correct but is incomplete, since we need to know composition of each refined gas, how much of each crude goes into each mix. A 2nd stab is "How to mix crude gas to produce refined gas?" This is right question, but is not yet in proper form for an algebraic model. Imagine you are manager of refinery. On 1 side you have all these tanks filled with crude gas, & on other side all empty tanks that will contain refined gas. In between: miles of pipes with valves that you control. What you really want to know is which valves to open & by how much to have exactly right mix. So proper question is "How much of each crude gas goes into each refined gas?"
– Câu hỏi cần trả lời là gì? Hãy hỏi câu hỏi này nhiều lần với độ chính xác ngày càng tăng. Câu hỏi đầu tiên là "Sản xuất bao nhiêu mỗi loại khí tinh chế?" Câu hỏi này đúng nhưng chưa đầy đủ, vì chúng ta cần biết thành phần của mỗi loại khí tinh chế, bao nhiêu mỗi loại dầu thô được đưa vào mỗi hỗn hợp. Câu hỏi thứ hai là "Làm thế nào để pha trộn khí thô để sản xuất khí tinh chế?" Đây là câu hỏi đúng, nhưng vẫn chưa ở dạng phù hợp cho 1 mô hình đại số. Hãy tưởng tượng bạn là người quản lý 1 nhà máy lọc dầu. 1 bên bạn có tất cả các thùng chứa đầy khí thô, & bên kia là tất cả các thùng rỗng sẽ chứa khí tinh chế. Ở giữa: hàng dặm đường ống có van do bạn điều khiển. Điều bạn thực sự muốn biết là mở van nào & bao nhiêu để có được hỗn hợp chính xác. Vậy câu hỏi đúng là "Lượng khí thô được đưa vào mỗi loại khí tinh chế là bao nhiêu?"

Key difference between *mixing* problems of preceding sect & this *blending* problem: previously we were told exact composition of products in terms of material (e.g., in each food, amount of each nutrient) while in problem considered here, composition of each product is 1 of answers sought.
– Sự khác biệt chính giữa các bài toán *trộn* của phần trước & bài toán *trộn* này: trước đây chúng ta được cho biết thành phần chính xác của sản phẩm về mặt vật liệu (ví dụ, trong mỗi loại thực phẩm, lượng của từng chất dinh dưỡng) trong khi trong bài toán được xem xét ở đây, thành phần của mỗi sản phẩm là 1 trong những câu trả lời được tìm kiếm.
Since we need to know how much of raw i goes into refined j, we are led to a 2D decision variable, say G_{ij}, where i is index of crude gas & j is index of refined gas. E.g., $G_{51} = 250$ will mean: there are 250 barrels of crude 5 going into mix of refined 1. Understand here: units will be barrels; it seems natural because prices are per barrels. Probably should also introduce auxiliary variables to help us model & present solution: total of each crude gas (sum of a row of G), say R_i, & total of each refined gas (sum of a column of G), say F_j. So we will have these non-constraining equaitons in model:

$$R_i = \sum_j G_{ij},\ \forall i,$$
$$F_j = \sum_i G_{ij},\ \forall j.$$

Note: by construction, $\sum_i R_i = \sum_j F_j$. I.e., total volume of crude used = total volume of refined products. Need not enforce this, though we need it. Can think of this as a "continuity" equation: it reflects: refining process does not lose product along way. This idea of continuity is a useful modeling idea. It will reappear in various guises throughout models we develop.
– Vì chúng ta cần biết bao nhiêu i thô được đưa vào j tinh chế, nên chúng ta được dẫn đến 1 biến quyết định 2D, chẳng hạn như G_{ij}, trong đó i là chỉ số của khí thô & j là chỉ số của khí tinh chế. Ví dụ: $G_{51} = 250$ sẽ có nghĩa là: có 250 thùng dầu thô 5 được đưa vào hỗn hợp của khí tinh chế 1. Hiểu ở đây: đơn vị sẽ là thùng; điều này có vẻ tự nhiên vì giá được tính theo thùng. Có lẽ cũng nên đưa vào các biến phụ trợ để giúp chúng ta mô hình hóa & trình bày giải pháp: tổng của mỗi loại khí thô (tổng của 1 hàng G), chẳng hạn như R_i, & tổng của mỗi loại khí tinh chế (tổng của 1 cột G), chẳng hạn như F_j. Vậy chúng ta sẽ có các phương trình không ràng buộc này trong mô hình:

$$R_i = \sum_j G_{ij},\ \forall i,$$
$$F_j = \sum_i G_{ij},\ \forall j.$$

Lưu ý: theo cấu trúc, $\sum_i R_i = \sum_j F_j$. Tức là, tổng thể tích dầu thô sử dụng = tổng thể tích sản phẩm tinh chế. Không cần thiết phải áp dụng điều này, mặc dù chúng ta cần nó. Có thể coi đây là 1 phương trình "liên tục": nó phản ánh: quá trình tinh chế không bị mất sản phẩm trong quá trình. Ý tưởng về tính liên tục này là 1 ý tưởng mô hình hữu ích. Nó sẽ xuất hiện trở lại dưới nhiều hình thức khác nhau trong các mô hình chúng ta phát triển.
Armed with these variables, can now easily model objective function. We are asked to maximize profits, hence difference between total sales (given price p_j for refined gas j) & costs (given cost c_i for crude gas i),

$$\max \sum_j F_j p_j - \sum_i R_i c_i.$$

– Được trang bị các biến này, giờ đây có thể dễ dàng mô hình hóa hàm mục tiêu. Chúng ta được yêu cầu tối đa hóa lợi nhuận, do đó chênh lệch giữa tổng doanh số (giá p_j cho khí tinh chế j) & chi phí (giá c_i cho khí thô i),

$$\max \sum_j F_j p_j - \sum_i R_i c_i.$$

Constraints come in multiple forms. Easy ones are, as in mixing problems, constraints on availability of each raw material. With our auxiliary variables, these are simple to express & can be included in range of defined variables or in a constraint

$$0 \le R_i \le u_i,\ \forall i.$$

Constraints on demand of refined gas (minimum &/or maximum/28) are just as simple:

$$a_j \le F_j \le b_j,\ \forall i.$$

Notice how our auxiliary variables help write down constraints. Having only our decision variables, constraints would have to be written w.r.t. column & row sums.
– Ràng buộc có nhiều dạng. Những ràng buộc dễ dàng, như trong các bài toán trộn, là ràng buộc về tính khả dụng của từng nguyên liệu thô. Với các biến phụ trợ của chúng ta, chúng rất dễ biểu diễn & có thể được bao gồm trong phạm vi các biến đã xác định hoặc trong 1 ràng buộc

$$0 \le R_i \le u_i,\ \forall i.$$

Ràng buộc về nhu cầu khí tinh chế (tối thiểu &/hoặc tối đa/28) cũng đơn giản như vậy:

$$a_j \leq F_j \leq b_j, \ \forall i.$$

Lưu ý cách các biến phụ trợ của chúng ta giúp viết ra các ràng buộc. Chỉ với các biến quyết định, các ràng buộc sẽ phải được viết theo tổng cột & hàng.

Only real complication of this problem refers to octane rating. Key here: assumption of linearity. To see how to model octane requirement, imagine a simple case: say mix 800 barrels of crude 1 with octane rating of 98 with 200 barrels of crude 2 with octane rating of 90. What is resulting octane rating? Since we have a total of 1000 barrels of refined, $\frac{800 \cdot 98 + 200 \cdot 90}{1000} = 96.4$. So, in general, need fraction of each crude that goes into a mix times its octane rating. Assuming O_i as octane rating of crude i & o_j octane rating of refined j, this leads us to

$$\sum_i O_i G_{ij} = F_j o_j, \ \forall j.$$

Now have an algebraic linear model. Translate it into executable code, as in Listing 2.4: Gasoline Blending Model. Will assume: data are entered in 2D arrays, exactly as in Tables 2.3–2.4, except for 1st columns, added for reference only.

– Biến chứng thực sự duy nhất của bài toán này liên quan đến chỉ số octan. Điểm mấu chốt ở đây: giả định tính tuyến tính. Để xem cách mô hình hóa yêu cầu về chỉ số octan, hãy tưởng tượng 1 trường hợp đơn giản: giả sử trộn 800 thùng dầu thô 1 có chỉ số octan là 98 với 200 thùng dầu thô 2 có chỉ số octan là 90. Chỉ số octan thu được là bao nhiêu? Vì chúng ta có tổng cộng 1000 thùng dầu tinh chế, nên $\frac{800 \cdot 98 + 200 \cdot 90}{1000} = 96,4$. Vì vậy, nhìn chung, cần phân số của mỗi loại dầu thô được đưa vào hỗn hợp nhân với chỉ số octan của nó. Giả sử O_i là chỉ số octan của dầu thô i & o_j là chỉ số octan của dầu tinh chế j, điều này dẫn đến

$$\sum_i O_i G_{ij} = F_j o_j, \ \forall j.$$

Giờ chúng ta có 1 mô hình tuyến tính đại số. Biên dịch nó thành mã thực thi, như trong Liệt kê 2.4: Mô hình Pha trộn Xăng. Giả sử: dữ liệu được nhập vào mảng 2D, chính xác như trong Bảng 2.3–2.4, ngoại trừ cột đầu tiên, được thêm vào chỉ để tham khảo.

At lines 3–5 declare some constants to access appropriate rows & columns of data. Constraints on range of each variable are entered not as constraints, but rather as a range on corresponding variables. Equations (2.2)–(2.3) are seen on 4 lines starting at 10.

– Ở dòng 3-5, hãy khai báo 1 số hằng số để truy cập các hàng & cột dữ liệu thích hợp. Các ràng buộc về phạm vi của mỗi biến được nhập không phải dưới dạng ràng buộc, mà là phạm vi của các biến tương ứng. Các phương trình (2.2)–(2.3) được hiển thị trên 4 dòng bắt đầu từ 10.

Blending equations are created on loop of line 14. Note since goal: achieve a certain octane level, might replace equality with an inequality, indicating: refined product has *at least* required octane level. This relaxes problem a little & allows optimization over a larger space. This might be required if, e.g., we did not have sufficient low octane crude gasolines available. Objective function (3 lines starting at 17) maximizes difference between selling price of refined product & cost of crude gas used. Executing this model with data above produces Table 2.5: Complete Solution to Blending Problem where bottom right number is profit: difference between sum of row Price & column Cost.

– Các phương trình pha trộn được tạo trên vòng lặp của dòng 14. Lưu ý rằng vì mục tiêu: đạt được 1 mức octan nhất định, nên có thể thay thế đẳng thức bằng 1 bất đẳng thức, cho biết: sản phẩm tinh chế có *ít nhất* mức octan cần thiết. Điều này giúp giải quyết vấn đề 1 chút & cho phép tối ưu hóa trên 1 không gian lớn hơn. Điều này có thể cần thiết nếu, ví dụ, chúng ta không có đủ xăng thô có chỉ số octan thấp. Hàm mục tiêu (3 dòng bắt đầu từ 17) tối đa hóa chênh lệch giữa giá bán sản phẩm tinh chế & chi phí khí thô đã sử dụng. Thực hiện mô hình này với dữ liệu trên sẽ tạo ra Bảng 2.5: Giải pháp hoàn chỉnh cho bài toán pha trộn, trong đó số dưới cùng bên phải là lợi nhuận: chênh lệch giữa tổng của hàng Giá & cột Chi phí.

∗ 2.2.2. Variations. While blending problems can be presented in various ways, they can all be handled in manner above. Decision variables should be 2D: sum in 1D & the other indicating total input material used & total output material produced. Finally, in addition to capacity & demand constraints, there should be at least 1 blending constraint satisfying a linearity assumption.

– Mặc dù các vấn đề pha trộn có thể được trình bày theo nhiều cách khác nhau, nhưng tất cả đều có thể được xử lý theo cách trên. Các biến quyết định nên là 2 chiều: tổng trong 1 chiều & chiều còn lại biểu thị tổng vật liệu đầu vào được sử dụng & tổng vật liệu đầu ra được sản xuất. Cuối cùng, ngoài các ràng buộc về công suất & nhu cầu, cần có ít nhất 1 ràng buộc pha trộn thỏa mãn giả định tuyến tính.

1 interesting variation: we might be asked to achieve > 1 characteristic. E.g., in addition to an octave level, we might also be given a certain concentration of sulfur in each of crude & asked to keep refined gas below a certain sulfur threshold. In this case, octane equation (2.4) will almost certainly need to be replaced by an inequality, ensuring a minimum octane level, & another similar inequality will ensure a maximum sulfur level. Assuming S_i as sulfur level of crude i & s_j as sulfur level of refined j, get

$$\sum_i O_i G_{ij} \leq F_j o_j, \ \sum_i S_i G_{ij} \geq F_j s_j, \ \forall j.$$

Reason for inequalities: unlikely for problem to have any feasible solution with exactly specified octane & sulfur levels. Reader might try to modify Listing 2.4 to verify this.

– 1 biến thể thú vị: chúng ta có thể được yêu cầu đạt được đặc tính > 1. Ví dụ, ngoài mức octave, chúng ta cũng có thể được cung cấp 1 nồng độ lưu huỳnh nhất định trong mỗi loại khí thô & được yêu cầu giữ khí tinh chế dưới 1 ngưỡng lưu huỳnh nhất định. Trong trường hợp này, phương trình octan (2.4) gần như chắc chắn sẽ cần được thay thế bằng 1 bất đẳng thức, đảm bảo mức octan tối thiểu, & 1 bất đẳng thức tương tự khác sẽ đảm bảo mức lưu huỳnh tối đa. Giả sử S_i là mức lưu huỳnh của dầu thô i & s_j là mức lưu huỳnh của khí tinh chế j, ta có

$$\sum_i O_i G_{ij} \leq F_j o_j, \ \sum_i S_i G_{ij} \geq F_j s_j, \ \forall j.$$

Lý do của bất đẳng thức: bài toán khó có thể có bất kỳ lời giải khả thi nào với mức octan & lưu huỳnh được chỉ định chính xác. Người đọc có thể thử sửa đổi Liệt kê 2.4 để kiểm tra điều này.

To help reader recognize underlying structure of blending problems, following is an instance we will revisit soon, with additional complexities.

– Để giúp người đọc nhận ra cấu trúc cơ bản của các vấn đề pha trộn, sau đây là 1 ví dụ mà chúng tôi sẽ sớm xem xét lại, với độ phức tạp bổ sung.

A very popular ingredient in junk food is manufactured by refining & blending various oils together. Oils come in 5 flavors O1 to O5 & measures of "hardness" as given in Table 2.6, where cost is in dollars per tons & hardness is measured in appropriate unit. Oils O1 & O2 can be refined at production facility A, which has a capacity of 200 tons per month, while O3, O4, & O5 can be refined at production facility B, which has a capacity of 250 tons per month. There is no loss of weight during refining process & you can ignore cost of process.

– 1 thành phần rất phổ biến trong đồ ăn vặt được sản xuất bằng cách tinh chế & pha trộn nhiều loại dầu khác nhau. Dầu có 5 loại từ O1 đến O5 & độ cứng được đo bằng đơn vị tương ứng trong Bảng 2.6, trong đó chi phí được tính bằng đô la/tấn & độ cứng được đo bằng đơn vị tương ứng. Dầu O1 & O2 có thể được tinh chế tại cơ sở sản xuất A, với công suất 200 tấn mỗi tháng, trong khi O3, O4, & O5 có thể được tinh chế tại cơ sở sản xuất B, với công suất 250 tấn mỗi tháng. Không có sự hao hụt trọng lượng nào trong quá trình tinh chế & bạn có thể bỏ qua chi phí quy trình.

Final product is obtained by mixing various amounts of 5 oils. It has a hardness restriction. Measured in same unit as given in table, it must lie between 3 & 6 units. Assumed: hardness blends linearly. I.e., if mix 10 tons of oil O1 with 20 tons of oil O2, blend will have a hardness rating of $\frac{10 \cdot 8.8 + 20 \cdot 6.1}{10+20}$. Final product sells for \$150 per ton. How should oils be refined & blended to maximize profit?

– Sản phẩm cuối cùng thu được bằng cách trộn 5 loại dầu với lượng khác nhau. Sản phẩm này có giới hạn độ cứng. Được đo bằng cùng đơn vị như trong bảng, sản phẩm phải nằm trong khoảng từ 3 đến 6 đơn vị. Giả sử: độ cứng được pha trộn tuyến tính. Ví dụ, nếu trộn 10 tấn dầu O1 với 20 tấn dầu O2, hỗn hợp sẽ có độ cứng là $\frac{10 \cdot 8.8 + 20 \cdot 6.1}{10+20}$. Sản phẩm cuối cùng được bán với giá \$150 mỗi tấn. Dầu nên được tinh chế & pha trộn như thế nào để tối đa hóa lợi nhuận?

◦ 2.3. Project Management. Project management, as is usually understood in context of optimization, refers to a set T of tasks, each with 2 properties:

1. A duration
2. A subset of T (possibly empty) of preceding tasks

Classic example is house construction: tasks include finding location, drawing plans, getting permits, breaking ground, laying foundations, building walls, installing plumbing, bribing inspectors, etc. Crucially, some tasks must be done before others: cannot build roof until raise walls. Main question under consideration: "When should each task start to minimize total project completion time?" I.e., when do we start each task to have house entirely built in shortest time possible? Also, if 1 task falls behind schedule, what is impact on all ulterior tasks & how do we reschedule them? Table 2.7: Example of Project Management Tasks is an instance of such a project & will use to illustrate a solution technique.

– Quản lý dự án, theo cách hiểu thông thường trong bối cảnh tối ưu hóa, đề cập đến 1 tập hợp T các nhiệm vụ, mỗi nhiệm vụ có 2 thuộc tính:

1. Thời lượng
2. 1 tập con của T (có thể rỗng) các nhiệm vụ trước đó

Ví dụ điển hình là xây dựng nhà: các nhiệm vụ bao gồm tìm địa điểm, vẽ sơ đồ, xin giấy phép, khởi công, đặt móng, xây tường, lắp đặt hệ thống ống nước, hối lộ thanh tra, v.v. Quan trọng là, 1 số nhiệm vụ phải được thực hiện trước những nhiệm vụ khác: không thể xây mái cho đến khi xây xong tường. Câu hỏi chính đang được xem xét: "Khi nào nên bắt đầu mỗi nhiệm vụ để giảm thiểu tổng thời gian hoàn thành dự án?" Tức là, khi nào chúng ta bắt đầu mỗi nhiệm vụ để hoàn thành việc xây dựng toàn bộ ngôi nhà trong thời gian ngắn nhất có thể? Ngoài ra, nếu 1 nhiệm vụ bị chậm tiến độ, điều gì sẽ ảnh hưởng đến tất cả các nhiệm vụ tiếp theo & làm thế nào để chúng ta lên lịch lại chúng? Bảng 2.7: Ví dụ về các Nhiệm vụ Quản lý Dự án là 1 ví dụ về 1 dự án như vậy & sẽ được sử dụng để minh họa 1 kỹ thuật giải quyết.

∗ 2.3.1. Constructing a Model. What we need to decide in this instance is how early to start each task, respecting precedence, to minimize total completion time. This suggests, as a decision variable, starting time of each task in same units as given durations. Assume a set T of tasks (corresponding to 1st column of Table 2.7) to declare our decision variables as $0 \leq t_i$, $\forall i \in T$. To ensure that precedence requirements are met, assume: we have, in addition to duration D_i (corresponding to 2nd column of Table 2.7), subsets $T_i \subset T$ of preceding tasks for each task i (corresponding to 3rd column of Table 2.7). Then need to lower bound starting times by

$$t_j + D_j \leq t_i, \ \forall j \in T_i, \ \forall i \in T.$$

Objective: minimize project completion time. This time would be starting time of last task plus its duration if tasks were all done sequentially. But they are likely not; we might be doing as many tasks in parallel as possible. Then how do we find completion time if we do not know last task, or if there is no single "last" task"?
– Điều chúng ta cần quyết định trong trường hợp này là bắt đầu mỗi tác vụ sớm như thế nào, tôn trọng thứ tự ưu tiên, để giảm thiểu tổng thời gian hoàn thành. Điều này gợi ý, như 1 biến quyết định, thời gian bắt đầu của mỗi tác vụ theo cùng đơn vị với thời lượng cho trước. Giả sử 1 tập T các tác vụ (tương ứng với cột thứ nhất của Bảng 2.7) để khai báo các biến quyết định của chúng ta là $0 \leq t_i, \forall i \in T$. Để đảm bảo các yêu cầu về thứ tự ưu tiên được đáp ứng, giả sử: ngoài thời lượng D_i (tương ứng với cột thứ hai của Bảng 2.7), chúng ta có các tập con $T_i \subset T$ gồm các tác vụ trước đó cho mỗi tác vụ i (tương ứng với cột thứ ba của Bảng 2.7). Sau đó, cần hạ thấp giới hạn thời gian bắt đầu bằng

$$t_j + D_j \leq t_i, \ \forall j \in T_i, \ \forall i \in T.$$

Mục tiêu: giảm thiểu thời gian hoàn thành dự án. Thời gian này sẽ bằng thời gian bắt đầu của nhiệm vụ cuối cùng cộng với thời gian thực hiện của nó nếu tất cả các nhiệm vụ được thực hiện tuần tự. Nhưng khả năng cao là không phải vậy; chúng ta có thể thực hiện càng nhiều nhiệm vụ song song càng tốt. Vậy thì làm thế nào để tìm thời gian hoàn thành nếu chúng ta không biết nhiệm vụ cuối cùng, hoặc nếu không có nhiệm vụ "cuối cùng" nào?
Introduce another variable t. Will constraint this t to be larger than, for each task, its starting time plus its duration. It will therefore be larger than completion time. & if add objective $\min t$ to set of constraints

$$t_i + D_i \leq t, \ \forall i \in T,$$

then t will, at optimality, be completion time, a condition that will hold no matter how many tasks we do in parallel.
– Giới thiệu 1 biến t khác. Sẽ ràng buộc t này phải lớn hơn, đối với mỗi tác vụ, thời gian bắt đầu cộng với thời gian thực hiện của nó. Do đó, nó sẽ lớn hơn thời gian hoàn thành. & nếu thêm mục tiêu $\min t$ vào tập các ràng buộc

$$t_i + D_i \leq t, \ \forall i \in T,$$

thì t, ở trạng thái tối ưu, sẽ là thời gian hoàn thành, 1 điều kiện sẽ đúng bất kể chúng ta thực hiện bao nhiêu tác vụ song song.
This is translated into an executable model in Listing 2.5: Project Management Model where we assume: data is given to us in Table D with same structure as Table 2.7: each row has a task identifier, a duration, & a set, possibly empty of preceding tasks.
– Điều này được dịch thành 1 mô hình thực thi trong Liệt kê 2.5: Mô hình quản lý dự án trong đó chúng ta giả sử: dữ liệu được cung cấp cho chúng ta trong Bảng D có cùng cấu trúc như Bảng 2.7: mỗi hàng có 1 mã định danh tác vụ, 1 khoảng thời gian, & 1 tập hợp, có thể không có tác vụ nào trước đó.
Line 4 computes a valid upper bound on times by adding all durations. This is clearly an overestimate but is fine to use in declaration of decision variables at line 5. Declare total completion time variable at line 6, which we use as an upper bound on all starting times plus duration at line 8. Finally, add precedence bounds at line 10. Results appear in Table 2.8: 1 Optimal Solution to Project Management Problem &, graphically, in Fig. 2.1: Graphical representation of example solution (nodes are times). Note: last ending time is total project completion time.
– Dòng 4 tính toán 1 giới hạn trên hợp lệ cho thời gian bằng cách cộng tất cả các khoảng thời gian. Đây rõ ràng là 1 ước tính quá cao nhưng vẫn ổn khi sử dụng trong khai báo các biến quyết định ở dòng 5. Khai báo biến thời gian hoàn thành tổng thể ở dòng 6, biến này được sử dụng làm giới hạn trên cho tất cả thời gian bắt đầu cộng với thời gian hoàn thành ở dòng 8. Cuối cùng, thêm các giới hạn ưu tiên ở dòng 10. Kết quả xuất hiện trong Bảng 2.8: 1 Giải pháp Tối ưu cho Bài toán Quản lý Dự án &, dưới dạng đồ họa, trong Hình 2.1: Biểu diễn đồ họa của giải pháp ví dụ (các nút là thời gian). Lưu ý: thời gian kết thúc cuối cùng là tổng thời gian hoàn thành dự án.
Note: all tasks could have started at any time after their required tasks have ended, & in fact, depending on solver used, solution might look rather different. Can see an example of an alternate solution in Table 2.9: An Alternate Optimal Solution to Project Management Problem. This situation of multiple optimal solutions offers us, as modelers, an opportunity to improve model. In this particular case, it might be useful to start all tasks as early as possible. This will not affect total completion time but might make project more practical & less prone to delays if some tasks' duration were poorly estimated.
– Lưu ý: tất cả các tác vụ có thể bắt đầu bất cứ lúc nào sau khi các tác vụ bắt buộc của chúng kết thúc, & trên thực tế, tùy thuộc vào trình giải được sử dụng, lời giải có thể trông khá khác nhau. Có thể xem ví dụ về 1 lời giải thay thế trong Bảng 2.9: 1 Lời Giải Tối Ưu Thay Thế cho Bài Toán Quản Lý Dự Án. Tình huống có nhiều lời giải tối ưu này mang đến cho chúng ta, với tư cách là người lập mô hình, cơ hội để cải thiện mô hình. Trong trường hợp cụ thể này, việc bắt đầu tất cả các tác vụ càng sớm càng tốt có thể hữu ích. Điều này sẽ không ảnh hưởng đến tổng thời gian hoàn thành nhưng có thể giúp dự án thực tế hơn & ít bị trì hoãn hơn nếu thời gian của 1 số tác vụ được ước tính kém.
Note finally: by looking at graphical representation, clear: subset of tasks 0,2,1,6,7,9 is critical in sense that if any of them are delayed, project completion time is delayed. On small projects, such a graphical representation is sufficient to identify critical tasks. On larger projects, it might be profitable to identify these tasks programmatically. See 1 way to compute critical paths in Sect. 4.4.3 in Chap. 4 when discuss longest paths.
– Cuối cùng, lưu ý: bằng cách xem biểu diễn đồ họa, hãy làm rõ: tập hợp con các tác vụ 0, 2, 1, 6, 7, 9 là quan trọng theo nghĩa là nếu bất kỳ tác vụ nào trong số chúng bị trì hoãn, thời gian hoàn thành dự án cũng bị trì hoãn. Đối với các dự án nhỏ, biểu diễn đồ họa như vậy là đủ để xác định các tác vụ quan trọng. Đối với các dự án lớn hơn, việc xác định các

tác vụ này bằng phương pháp lập trình có thể mang lại lợi ích. Xem 1 cách tính đường dẫn quan trọng trong Mục 4.4.3 của Chương 4 khi thảo luận về đường dẫn dài nhất.

∗ 2.3.2. Variations. Displayed 2 possible solutions to problem in Figs. 2.1–2.2. Alternate might be preferable for practical reason. How can we ensure: among all solutions that minimize total completion time, choose a solution that starts all tasks as early as possible? 1 way: minimize sum of starting times. I.e., replace objective function by

```
s.Minimize(sum(t[i] for i in range(n)))
```

In cases like this, optimizers talk of *multiple objectives*. In general, these might be independent, or worse, contradictory. But in our project management situation, objectives (minimizing completion time & starting all tasks as early as possible) are coherent. Note new optimal value of model is neither interesting nor useful. Need to inspect `Total` variable to give us completion time.

– Hiển thị 2 giải pháp khả thi cho bài toán trong Hình 2.1–2.2. Giải pháp thay thế có thể được ưu tiên hơn vì lý do thực tế. Làm thế nào chúng ta có thể đảm bảo: trong số tất cả các giải pháp giảm thiểu tổng thời gian hoàn thành, hãy chọn 1 giải pháp bắt đầu tất cả các tác vụ càng sớm càng tốt? Cách 1: giảm thiểu tổng thời gian bắt đầu. Ví dụ, thay thế hàm mục tiêu bằng

```
s.Minimize(sum(t[i] for i in range(n)))
```

Trong những trường hợp như thế này, các trình tối ưu hóa nói về *nhiều mục tiêu*. Nhìn chung, chúng có thể độc lập, hoặc tệ hơn, mâu thuẫn. Nhưng trong tình huống quản lý dự án của chúng ta, các mục tiêu (giảm thiểu thời gian hoàn thành & bắt đầu tất cả các tác vụ càng sớm càng tốt) là nhất quán. Lưu ý rằng giá trị tối ưu mới của mô hình không thú vị cũng không hữu ích. Cần kiểm tra biến `Total` để cho chúng ta thời gian hoàn thành.

· 2.3.2.1. Minimax Problems. Technique we used for project management can be used more generally whenever we face a *minimax* problem. This is a problem where we want to minimize maximum of some set of functions. E.g., assume we want to find optimal x for

$$\min_x \max_{i \in T} \sum_j a_{ij} x_j.$$

This is handled by introducing a new variable, say t, along with objective $\min t$ & constraints

$$\sum_j a_{ij} x_j \le t, \ \forall i \in T.$$

Corresponding *maximin* problem is handled similarly. Note: related *maximax & minimin* are considerably more difficult to handle. Revisit those in Sect. 7.2.4 in Chap. 7.

– Kỹ thuật chúng tôi sử dụng để quản lý dự án có thể được sử dụng rộng rãi hơn bất cứ khi nào chúng ta gặp phải bài toán *minimax*. Đây là bài toán mà chúng ta muốn tối thiểu hóa giá trị cực đại của 1 tập hợp hàm nào đó. Ví dụ: giả sử chúng ta muốn tìm x tối ưu cho

$$\min_x \max_{i \in T} \sum_j a_{ij} x_j.$$

Điều này được xử lý bằng cách đưa vào 1 biến mới, chẳng hạn t, cùng với mục tiêu $\min t$ & các ràng buộc

$$\sum_j a_{ij} x_j \le t, \forall i \in T.$$

Bài toán *maximin* tương ứng được xử lý tương tự. Lưu ý: các *maximax & minimin* liên quan khó xử lý hơn đáng kể. Xem lại những điều đó trong Mục 7.2.4 của Chương 7.

· 2.3.2.2. Absolute Value Problems. Essentially same approach can also be used for some nonlinear functions, e.g., those involving absolute values. Say we seek optimal x for $\min_x |\sum_j c_j x_j|$. Due to def of absolute value function, we can use same $\min t$ objective along with constraints

$$\sum_j c_j x_j \le t, \ -\sum_j c_j x_j \le t, \ \forall i \in T.$$

Illustrate some nontrivial applications of this technique in Sect. 3.2 in Chap. 3.

– Về cơ bản, cách tiếp cận tương tự cũng có thể được sử dụng cho 1 số hàm phi tuyến tính, ví dụ như các hàm liên quan đến giá trị tuyệt đối. Giả sử chúng ta tìm kiếm x tối ưu cho $\min_x |\sum_j c_j x_j|$. Do định nghĩa của hàm giá trị tuyệt đối, chúng ta có thể sử dụng cùng 1 mục tiêu $\min t$ cùng với các ràng buộc.

$$\sum_j c_j x_j \le t, \ -\sum_j c_j x_j \le t, \ \forall i \in T.$$

Minh họa 1 số ứng dụng không tầm thường của kỹ thuật này trong Mục 3.2 của Chương 3.

- 2.4. Multi-Stage Models. In life, decisions at 1 stage often influence decisions at a later stage. Same holds for more pedestrian situations. E.g., consider a warehouse: what it contains at end of a month surely should influence what is ordered at beginning of following month. In a certain sense there is little that is new in these multi-stage models except: we have to be careful to properly set up continuity from 1 stage to next.

  – Mô hình Đa giai đoạn. Trong cuộc sống, các quyết định ở giai đoạn đầu thường ảnh hưởng đến các quyết định ở giai đoạn sau. Điều này cũng đúng với những tình huống đơn giản hơn. Ví dụ, hãy xem xét 1 nhà kho: những gì nó chứa vào cuối tháng chắc chắn sẽ ảnh hưởng đến những gì được đặt hàng vào đầu tháng tiếp theo. Theo 1 nghĩa nào đó, không có gì mới trong các mô hình đa giai đoạn này ngoại trừ: chúng ta phải cẩn thận thiết lập tính liên tục hợp lý từ giai đoạn này sang giai đoạn tiếp theo.

  To illustrate, revisit blending problem. to multiple targets, prices, & costs, will add a planning horizon of many months. This will exemplify stages. This problem will require all tricks & techniques you have seen so far (& then some). It forms a comprehensive review of current chap.

  – Để minh họa, hãy xem lại bài toán pha trộn. Việc kết hợp nhiều mục tiêu, giá cả, & chi phí sẽ làm tăng thêm thời gian lập kế hoạch lên nhiều tháng. Điều này sẽ minh họa các giai đoạn. Bài toán này sẽ yêu cầu tất cả các thủ thuật & kỹ thuật bạn đã thấy cho đến nay (& thậm chí còn hơn thế nữa). Nó tạo nên 1 bản tóm tắt toàn diện về chương hiện tại.

  * 2.4.1. Problem Instance. Soap is manufactured by refining & blending various oils together. Oils come in various flavors (apricot, avocado, canola, coconut, etc.) & each coil contains multiple fatty acids (lauric, linoleic, oleic, etc.) in various proportions, see Table 2.10: Example of Oils (Oi) with Their Acid Content (Aj). According to properties of soap one is creating (cleaning power, lather production, dryness of skin, etc.) one targets final proportions of fatty acids to be in certain ranges by blending oils appropriately. E.g., will target our soap to have acid contents in ranges of Table 2.11: Fatty Acid Content Targets.

    – Xà phòng được sản xuất bằng cách tinh chế & pha trộn nhiều loại dầu khác nhau. Dầu có nhiều hương vị khác nhau (mơ, bơ, cải dầu, dừa, v.v.) & mỗi cuộn chứa nhiều axit béo (lauric, linoleic, oleic, v.v.) theo các tỷ lệ khác nhau, xem Bảng 2.10: Ví dụ về các loại dầu (Oi) với hàm lượng axit (Aj) của chúng. Tùy thuộc vào đặc tính của xà phòng mà người ta tạo ra (khả năng làm sạch, tạo bọt, độ khô da, v.v.), người ta nhắm mục tiêu tỷ lệ axit béo cuối cùng nằm trong 1 phạm vi nhất định bằng cách pha trộn các loại dầu 1 cách thích hợp. Ví dụ: chúng ta sẽ nhắm mục tiêu xà phòng của mình có hàm lượng axit trong phạm vi Bảng 2.11: Mục tiêu về hàm lượng axit béo.

    Here is an additional twist, relative to periods. Will be planning for a certain number of months. Each oil may be purchased for immediate delivery or bought on futures market for delivery in a later month. Price of each oil in each of months is given in Table 2.12: Cost of Oils in Dollars per Ton Over Planning Horizon in dollars per ton. Possible to store up to 1000 tons of oil for later use (any combination of oils) but there is a holding cost of $5 per ton per month. Finally, must satisfy a demand of 5000 tons of soap per month. This demand drives model.

    – Đây là 1 điểm khác biệt so với các giai đoạn. Sẽ lập kế hoạch cho 1 số tháng nhất định. Mỗi loại dầu có thể được mua để giao ngay hoặc mua trên thị trường tương lai để giao vào tháng sau. Giá của mỗi loại dầu trong mỗi tháng được thể hiện trong Bảng 2.12: Chi phí dầu tính bằng đô la/tấn theo thời gian kế hoạch tính bằng đô la/tấn. Có thể lưu trữ tới 1000 tấn dầu để sử dụng sau (bất kỳ sự kết hợp nào của các loại dầu) nhưng chi phí lưu kho là $5/tấn/tháng. Cuối cùng, phải đáp ứng nhu cầu 5000 tấn xà phòng mỗi tháng. Nhu cầu này chi phối mô hình.

    At beginning of planning horizon, we have some oils in inventory, as illustrated in Table 2.13: Initial Inventory in Tons. How should oils be refined & blended every month to minimize cost?

    – Vào đầu thời gian lập kế hoạch, chúng tôi có 1 số loại dầu tồn kho, như minh họa trong Bảng 2.13: Hàng tồn kho ban đầu (Tấn). Dầu nên được tinh chế & pha trộn như thế nào mỗi tháng để giảm thiểu chi phí?

  * 2.4.2. Constructing a Model.

    · 2.4.2.1. Decision variables. Question to answer is "How should various oils be blended every month?" I.e., need to identify how much of each oil goes into final blend during each month. This is a good start but clearly not enough. E.g., can blend from oil we buy & from oil we have in inventory.

    – Câu hỏi cần trả lời là "Nên pha trộn các loại dầu khác nhau như thế nào mỗi tháng?" Tức là, cần xác định lượng dầu cần pha trộn cuối cùng trong mỗi tháng. Đây là 1 khởi đầu tốt nhưng rõ ràng là chưa đủ. Ví dụ, có thể pha trộn từ dầu chúng ta mua & từ dầu chúng ta có trong kho.

    So we need to distinguish these 2 quantities. Moreover, may decide to buy for storage (because prices are about to go up) so we also need to know how much we can store. This suggests at least 3 decision variables for each oil ($O = \{0, 1, 2, \ldots, n_0\}$ will be set of oils), & for each month ($M = \{0, 1, 2, \ldots, n_m\}$ is set of months). Interpretation is x_{ij} will be number of tons of oil i bought during month j; y_{ij}: number of tons blended into our soap; & z_{ij} is number of tons held at beginning of month. Note: have a choice here to have variable represent amount at beginning or at end of period. Either is acceptable but it must be clear in model which one is chosen because it affects constraints. A typical mistake in a multi-period model: have some constraints assume that a variable represents a quantity at start of period while some other constraints assume end. Model may run, but solution will be nonsensical. Since we are given quantities in storage at beginning of planning period, having a variable represent quantity held at beginning means: we can easily initialize it with given data.

    – Vì vậy, chúng ta cần phân biệt 2 số lượng này. Hơn nữa, có thể quyết định mua để lưu trữ (vì giá sắp tăng) nên chúng ta cũng cần biết mình có thể lưu trữ bao nhiêu. Điều này cho thấy ít nhất 3 biến quyết định cho mỗi loại dầu ($O = \{0, 1, 2, \ldots, n_0\}$ sẽ là tập hợp các loại dầu), & cho mỗi tháng ($M = \{0, 1, 2, \ldots, n_m\}$ là tập hợp các tháng). Diễn giải là x_{ij} sẽ là số tấn dầu i đã mua trong tháng j; y_{ij}: số tấn được pha trộn vào xà phòng của chúng ta; & z_{ij} là số tấn được giữ vào đầu tháng. Lưu ý: có 1 lựa chọn ở đây để có biến biểu diễn số lượng vào đầu hoặc cuối kỳ. Cả hai đều

được chấp nhận nhưng mô hình phải làm rõ lựa chọn nào vì nó ảnh hưởng đến các ràng buộc. 1 lỗi điển hình trong mô hình nhiều kỳ: có 1 số ràng buộc giả định rằng 1 biến biểu diễn số lượng vào đầu kỳ trong khi 1 số ràng buộc khác giả định rằng kết thúc kỳ. Mô hình có thể chạy, nhưng giải pháp sẽ vô nghĩa. Vì chúng ta được cung cấp số lượng hàng tồn kho vào đầu kỳ lập kế hoạch, việc có 1 biến đại diện cho số lượng hàng tồn kho lúc đầu có nghĩa là: chúng ta có thể dễ dàng khởi tạo nó với dữ liệu cho sẵn.
Probably will need to know how much soap we are producing each month. This is not, strictly speaking, essential to problem as formulated, but it may make presentation of solution & maybe formulation of some constraints much simpler. As usual, it helps to introduce auxiliary variables to clear up some statements. To tally total production per month t_j, $\forall j \in M$.

· 2.4.2.2. Constraints. Tackle continuity constraints. Need to specify for each oil & for each month (but the last) how inventory fluctuates, so

$$z_{ij} + x_{ij} - y_{ij} = z_{i,j+1},\ \forall i \in O,\ \forall j \in M \backslash \{n_m\}.$$

I.e., this is what is held at beginning of month plus what we buy minus what we blend forms new inventory.
– Giải quyết các ràng buộc về tính liên tục. Cần chỉ rõ cho mỗi loại dầu & cho mỗi tháng (trừ tháng cuối cùng) mức độ biến động của hàng tồn kho, vì vậy

$$z_{ij} + x_{ij} - y_{ij} = z_{i,j+1}, \forall i \in O, \forall j \in M \backslash \{n_m\}.$$

Tức là, đây là lượng hàng tồn kho đầu tháng cộng với lượng hàng mua trừ đi lượng hàng pha trộn tạo thành hàng tồn kho mới.
Have a minimum & a maximum storage capacity at each month of total amount of oil, or

$$C_{\min} \leq \sum_i z_{ij} \leq C_{\max},\ \forall j \in M.$$

Now comes blending constraint, or rather constraints, since need to target a number of fatty acids. To help formulation, extract total production

$$t_j = \sum_i y_{ij},\ \forall j \in M.$$

Assume for each acid $k \in A$, we have a target range $[l_k, u_k]$ & each oil $i \in O$, a percentage p_{ik} of required acid. Since final product for each acid must fall in a certain range, should have 2 constraints: 1 for low end & 1 for high end of interval. I.e.,

$$\sum_i y_{ij} p_{ik} \geq l_k t_j,\ \sum_i y_{ij} p_{ik} \leq u_k t_j,\ \forall k \in A,\ \forall j \in M.$$

These constraints could be written without production variables t_j but would be more cumbersome & difficult to read. Finally, need to satisfy demand. This is simple, assuming a demand of D_j at each month j, $t_j \geq D_j$, $\forall j \in M$.
– Có dung lượng lưu trữ tối thiểu & tối đa tại mỗi tháng của tổng lượng dầu, hoặc

$$C_{\min} \leq \sum_i z_{ij} \leq C_{\max}, \forall j \in M.$$

Bây giờ đến ràng buộc pha trộn, hay đúng hơn là ràng buộc, vì cần nhắm mục tiêu đến 1 số axit béo. Để hỗ trợ công thức, hãy trích xuất tổng sản lượng

$$t_j = \sum_i y_{ij}, \forall j \in M.$$

Giả sử đối với mỗi axit $k \in A$, chúng ta có 1 khoảng mục tiêu $[l_k, u_k]$ & mỗi loại dầu $i \in O$, 1 phần trăm p_{ik} axit cần thiết. Vì sản phẩm cuối cùng của mỗi axit phải nằm trong 1 khoảng nhất định, nên cần có 2 ràng buộc: 1 cho giới hạn thấp & 1 cho giới hạn cao của khoảng. Tức là,

$$\sum_i y_{ij} p_{ik} \geq l_k t_j,\ \sum_i y_{ij} p_{ik} \leq u_k t_j,\ \forall k \in A,\ \forall j \in M.$$

Các ràng buộc này có thể được viết mà không cần biến sản xuất t_j nhưng sẽ phức tạp hơn & khó đọc. Cuối cùng, cần phải đáp ứng nhu cầu. Điều này rất đơn giản, giả sử nhu cầu là D_j tại mỗi tháng j, $t_j \geq D_j$, $\forall j \in M$.

· 2.4.2.3. Objective Function. Objective: minimize costs, comprised of varying oil costs at each month plus fixed storage cost of coils we keep in inventory. Therefore

$$\sum_i \sum_j x_{ij} P_{ij} + \sum_i \sum_j z_{ij} p.$$

This type of objective (fixed plus variable cost) appears regularly in business-type problems. Will see this again when considering facility location to service customer demands. Decision to build incurs a fixed cost. Servicing of various customers is a variable cost.

– Mục tiêu: giảm thiểu chi phí, bao gồm chi phí dầu biến động hàng tháng cộng với chi phí lưu kho cố định của các cuộn dây chúng tôi lưu kho. Do đó

$$\sum_i \sum_j x_{ij} P_{ij} + \sum_i \sum_j z_{ij} p.$$

Loại mục tiêu này (chi phí cố định cộng với chi phí biến đổi) thường xuyên xuất hiện trong các bài toán kinh doanh. Chúng ta sẽ thấy điều này 1 lần nữa khi xem xét vị trí cơ sở để phục vụ nhu cầu của khách hàng. Quyết định xây dựng sẽ phát sinh chi phí cố định. Việc phục vụ nhiều khách hàng khác nhau là 1 chi phí biến đổi.

· 2.4.2.4. Executable Model. Now translate this into executable code as shown in Listing 2.6: Multi-Period Blending Model. There is a fair amount of data to pass in. Assume arrays `Part` as in Table 2.10, `Target` as in Table 2.11, `Cost` as in Table 2.12, & `Inventory` as in Table 2.13 in addition to 3 parameters: `D` in tons for demand, `SC` in dollars per ton for storage cost, & `SL` in tons for minimum & maximum to hold in inventory.

– Bây giờ hãy dịch đoạn mã này thành mã thực thi như được hiển thị trong Liệt kê 2.6: Mô hình Pha trộn Đa kỳ. Có 1 lượng dữ liệu khá lớn cần truyền vào. Giả sử các mảng `Phần` như trong Bảng 2.10, `Mục tiêu` như trong Bảng 2.11, `Chi phí` như trong Bảng 2.12, & `Hàng tồn kho` như trong Bảng 2.13 cùng với 3 tham số: `D` tính bằng tấn cho nhu cầu, `SC` tính bằng đô la trên mỗi tấn cho chi phí lưu trữ, & `SL` tính bằng tấn cho mức tối thiểu & tối đa cần lưu trữ trong kho.

From line 5–11 declare variables but only 1st 3 are true decision variables. All others are artificially introduced either to help us state constraints (for 8 & 11) or to help us display some details of resulting solutions. They will not affect running time of solver in any appreciable manner but will make our life easier.

– Từ dòng 5-11, hãy khai báo các biến nhưng chỉ 3 biến đầu tiên là biến quyết định đúng. Tất cả các biến còn lại được đưa vào 1 cách giả tạo để giúp chúng ta xác định các ràng buộc (cho 8 & 11) hoặc để giúp chúng ta hiển thị 1 số chi tiết về các giải pháp kết quả. Chúng sẽ không ảnh hưởng đáng kể đến thời gian chạy của trình giải nhưng sẽ giúp mọi thứ dễ dàng hơn.

At line 12 set `Hold` variable to contain what is known to be in inventory at start of planning period. Large loop starting at line 14 will set all constraints since they have identical structure for each month & we have declared our variables to be arrays indexed by month.

– Ở dòng 12, hãy đặt biến `Hold` để chứa những gì được biết là có trong kho khi bắt đầu kỳ lập kế hoạch. Vòng lặp lớn bắt đầu từ dòng 14 sẽ thiết lập tất cả các ràng buộc vì chúng có cấu trúc giống hệt nhau cho mỗi tháng & chúng ta đã khai báo các biến của mình là các mảng được lập chỉ mục theo tháng.

Line 15 sets artificial variable `Prod` to be sum of blended oils. This is not really a constraint, but rather a simplifying trick. If repeat some calculations in a model as here `sum(Blnd[i][j] for i in range(nO))` we should consider introducing an artificial variable. Assuming a decent solver, it will cost nothing & is likely to help. 1 of principles of programming (& modeling) is "Do not repeat yourself".

– Dòng 15 đặt biến nhân tạo `Prod` là tổng của các loại dầu pha trộn. Đây không hẳn là 1 ràng buộc, mà là 1 mẹo đơn giản hóa. Nếu lặp lại 1 số phép tính trong mô hình như ở đây `sum(Blnd[i][j] for i in range(nO))`, chúng ta nên cân nhắc việc đưa vào 1 biến nhân tạo. Giả sử có 1 trình giải tốt, việc này sẽ không tốn kém gì & có thể hữu ích. 1 trong những nguyên tắc của lập trình (& mô hình hóa) là "Đừng lặp lại chính mình".

Use this `Prod` variable immediately after, at line 16 to ensure we satisfy demand. If this demand is a scalar, we set it identically for each month, but it could be an array index by month.

– Sử dụng biến `Prod` này ngay sau đó, tại dòng 16 để đảm bảo chúng ta đáp ứng được nhu cầu. Nếu nhu cầu này là 1 số vô hướng, chúng ta sẽ đặt nó giống hệt nhau cho mỗi tháng, nhưng nó có thể là 1 chỉ mục mảng theo từng tháng.

Code starting with `if` on line 17 implements continuity requirement described in (2.5). Ensure what we buy & what we have on hand at beginning of month equals what we blend & what we store for next month. Conditional is to avoid setting a constraint on a month past planning horizon.

– Mã bắt đầu bằng `if` trên dòng 17 thực hiện yêu cầu về tính liên tục được mô tả trong (2.5). Đảm bảo những gì chúng ta mua & những gì chúng ta có sẵn vào đầu tháng bằng với những gì chúng ta pha trộn & những gì chúng ta dự trữ cho tháng tiếp theo. Điều kiện là để tránh đặt ràng buộc cho 1 tháng sau thời hạn lập kế hoạch.

Lines 20–21 ensure bounds on oils we keep in inventory. Loop starting at line 22 1st defines our auxiliary `Acid` variable to ease formulation of blending constraints stated on following 2 lines, which correspond to (2.6)–(2.7). Acid, indexed by ordinal of fatty acid k & of month j under consideration, is summed over all oils of quantity blended with oil's percentage of acid k. This quantity, divided by total blended, will be percentage that must fall within required range.

– Dòng 20–21 đảm bảo giới hạn về lượng dầu chúng ta lưu trữ trong kho. Vòng lặp bắt đầu từ dòng 22, đầu tiên xác định biến phụ `Acid` của chúng ta để dễ dàng xây dựng các ràng buộc pha trộn được nêu trên 2 dòng sau, tương ứng với (2.6)–(2.7). Axit, được lập chỉ mục theo thứ tự của axit béo k & của tháng j đang xem xét, được cộng trên tất cả các loại dầu có khối lượng pha trộn với tỷ lệ phần trăm axit k của dầu. Khối lượng này, chia cho tổng khối lượng pha trộn, sẽ là tỷ lệ phần trăm phải nằm trong phạm vi yêu cầu.

Finally, 4 lines starting at 26 set artificial variables that will hold costs of purchasing & holding at each period & then sum them to construct objective function which we will minimize.

– Cuối cùng, 4 dòng bắt đầu từ 26 đặt các biến nhân tạo sẽ giữ chi phí mua & giữ ở mỗi kỳ & sau đó cộng chúng lại để xây dựng hàm mục tiêu mà chúng ta sẽ tối thiểu hóa.

Since this model is of a certain complexity, caller should examine return code of solver. It needs to be 0 for solution to be optimal. Most frequent nonzero return status will be for infeasibility. This may occur for a number of reasons, most likely of which is that there is no combination of oil that will achieve our target fatty acid content.

– Vì mô hình này có độ phức tạp nhất định, người gọi nên kiểm tra mã trả về của trình giải. Mã trả về phải bằng 0 để

đạt được giải pháp tối ưu. Trạng thái trả về khác 0 thường gặp nhất là do không khả thi. Điều này có thể xảy ra vì 1 số lý do, trong đó có thể là do không có sự kết hợp dầu nào đạt được hàm lượng axit béo mục tiêu.

Results of a run with all above data is displayed in Table 2.14: Multi-Period Blending Results. it displays everything we need to know. 1st set of lines, to be sent to Purchasing, specify how much of each oil to buy per month. Next set of lines, to be sent to Manufacturing, describe exact recipe of blending to do each month. Notice: soap is created from different oils in each month to achieve minimal cost. Next set of lines, to be sent to Bean Counters, describes inventory, product costs, & storage costs at each month. & finally, we can send to Quality Control last set of lines, indicating actual percentages of fatty acids achieved by blending recipe.

– Kết quả của 1 lần chạy với tất cả dữ liệu trên được hiển thị trong Bảng 2.14: Kết quả pha trộn nhiều kỳ. Nó hiển thị mọi thông tin chúng ta cần biết. Bộ dòng đầu tiên, được gửi đến Bộ phận Mua hàng, chỉ định số lượng từng loại dầu cần mua mỗi tháng. Bộ dòng tiếp theo, được gửi đến Bộ phận Sản xuất, mô tả công thức pha trộn chính xác cần thực hiện mỗi tháng. Lưu ý: xà phòng được tạo ra từ các loại dầu khác nhau trong mỗi tháng để đạt được chi phí tối thiểu. Bộ dòng tiếp theo, được gửi đến Bộ phận Kế toán, mô tả hàng tồn kho, chi phí sản phẩm, & chi phí lưu kho tại mỗi tháng. & cuối cùng, chúng ta có thể gửi đến Bộ phận Kiểm soát Chất lượng bộ dòng cuối cùng, cho biết tỷ lệ phần trăm thực tế của axit béo đạt được bằng công thức pha trộn.

Main point of this model: present complexity of real models along with some tricks on managing this complexity at model level. A 2nd point: highlight some of advantages of modeling in Python instead of in specialized modeling languages.

– Điểm chính của mô hình này: trình bày độ phức tạp của các mô hình thực tế cùng với 1 số mẹo để quản lý độ phức tạp này ở cấp độ mô hình. Điểm thứ hai: nêu bật 1 số ưu điểm của việc mô hình hóa bằng Python so với các ngôn ngữ mô hình hóa chuyên biệt.

* 2.4.3. Variations. There are an infinite number of variations of such a complex model.
  - Demand could vary at each month, as shown in Table 2.14.
  - Instead of satisfying some demand, we may be asked to maximize profit. In this case, need to know price of final product, which of course may change at each month.
  - Inventory levels may be stated in terms of each oil instead of aggregate quantities.
  - There may be uncertainty in fatty acid content of certain oils.
  - Có vô số biến thể của 1 mô hình phức tạp như vậy.
  - Nhu cầu có thể thay đổi hàng tháng, như thể hiện trong Bảng 2.14.
  - Thay vì đáp ứng 1 số nhu cầu, chúng ta có thể được yêu cầu tối đa hóa lợi nhuận. Trong trường hợp này, cần biết giá của sản phẩm cuối cùng, tất nhiên giá này có thể thay đổi hàng tháng.
  - Mức tồn kho có thể được thể hiện theo từng loại dầu thay vì tổng số lượng.
  - Có thể có sự không chắc chắn về hàm lượng axit béo trong 1 số loại dầu nhất định.

∘ 2.5. Pattern Classification. Classification is currently 1 of most successful applications of software to tasks that were, not so long ago, privilege of human intellect. E.g., software decides if an email is legitimate or spam, whether a biopsied cell is malignant or benign, & whether company should offer you an interview of let your resume rot in great bit bucket in sky.

– Phân loại hiện là 1 trong những ứng dụng thành công nhất của phần mềm vào các nhiệm vụ mà trước đây chỉ là đặc quyền của trí tuệ con người. Ví dụ, phần mềm quyết định xem 1 email là hợp pháp hay thư rác, liệu 1 tế bào được sinh thiết là ác tính hay lành tính, & liệu công ty có nên mời bạn phỏng vấn hay để hồ sơ của bạn thối rữa trong 1 cái xô lớn trên bầu trời.

Look at 1 of 1st effective techniques for binary classification of data. Example is contrived because I want to draw pictures to guide intuition, but code we will write is applicable in a wide variety of cases.

– Hãy xem xét 1 trong những kỹ thuật hiệu quả đầu tiên để phân loại dữ liệu nhị phân. Ví dụ này được thiết kế vì tôi muốn vẽ hình ảnh để hướng dẫn trực giác, nhưng đoạn mã chúng ta sẽ viết có thể áp dụng trong nhiều trường hợp khác nhau.

Imagine trying to automate classification of cells as malignant or benign based on 2 measures: area & perimeter. Those features are measured automatically from a picture of cell under a microscope. Process starts with a collection of such cells, divided by an expert into 2 groups. These groups form what is known as training set for our software. After we have "trained" our software, we will feed it new data, that has not been seen by an expert, & it will decide in which group cell falls. I.e., it will classify cell as malignant or benign. This process is real & used in laboratories all over world. Major simplification I am making here: many more than 2 features are used in practice.

– Hãy tưởng tượng việc cố gắng tự động phân loại tế bào là ác tính hay lành tính dựa trên 2 phép đo: diện tích & chu vi. Các đặc điểm này được đo tự động từ hình ảnh tế bào dưới kính hiển vi. Quá trình bắt đầu với 1 tập hợp các tế bào như vậy, được 1 chuyên gia chia thành 2 nhóm. Các nhóm này tạo thành cái được gọi là tập huấn luyện cho phần mềm của chúng tôi. Sau khi "huấn luyện" phần mềm, chúng tôi sẽ cung cấp cho nó dữ liệu mới, chưa được chuyên gia nhìn thấy, & nó sẽ quyết định tế bào thuộc nhóm nào. Tức là, nó sẽ phân loại tế bào là ác tính hay lành tính. Quá trình này là có thật & được sử dụng trong các phòng thí nghiệm trên toàn thế giới. Tôi đang đơn giản hóa rất nhiều ở đây: nhiều hơn 2 đặc điểm được sử dụng trong thực tế.

Consider as an example cell features plotted in Fig. 2.3: Cell data & separation hyperplane with perimeter on x-axis & radius on y-axis. See: 2 classes can be separated by a line. Our task: discover that line. Of course, there are a number of valid lines but, as a 1st attempt, any line separating 2 classes will do.

– Hãy xem xét ví dụ về các đặc điểm ô được vẽ trong Hình 2.3: Siêu mặt phẳng phân tách dữ liệu ô với chu vi trên trục x & bán kính trên trục y. Xem: 2 lớp có thể được phân tách bằng 1 đường thẳng. Nhiệm vụ của chúng ta: tìm ra đường thẳng đó. Tất nhiên, có 1 số đường thẳng hợp lệ, nhưng, như lần thử đầu tiên, bất kỳ đường thẳng nào phân tách 2 lớp đều được.

* 2.5.1. Constructing a Model. Algebraically, a line is an equation of the form $a_1x_1 + a_2x_2 = a_0$ for some fixed coefficient a_1, a_2, a_0. Or, in dimension n, we call it a hyperplane & it has an equation of $\sum_{i=1}^{n} a_i x_i = a_0$. What does it mean for a particular point x to be on 1 side or other of line? I.e., either $a_1x_1 + a_2x_2 < a_0$ or $a_1x_1 + a_2x_2 > a_0$. These strict inequalities can be scaled to increase gap by any amount. Can therefore simplify our task to identifying a vector a s.t., for every point x' in class A, we have $\sum_i a_i x'_i \geq a_0 + 1$ & that, for every point x'' in class B, we have $\sum_i a_i x''_i \leq a_0 - 1$. Introduce a positive variable for each of data points, say y'_i for each point of class A & y''_i for each of class B. Now inequality $\sum_i a_i x_i \geq a_0 + 1$ can be enforced by requiring

$$y' \geq a_0 + 1 - \sum_i a_i x'_i,\ y' \geq 0,$$

& minimizing y' to 0. Algebra is symmetric for points of class B. All in all, we are led to following optimization problem:

$$\min \sum_{i \in A} y'_i + \sum_{i \in B} y''_i \text{ subject to } \begin{cases} y' \geq a_0 + 1 - \sum_i a_i x'_i, \\ y'' \geq \sum_i a_i x''_i - a_0 + 1, \\ y', y'' \geq 0. \end{cases}$$

1 characteristic of this model: if optimal objective value is 0, we have a hyperplane correctly separating training set into malignant cells & benign cells. But if value is nonzero, i.e., set is not separable by a hyperplane & so more complex techniques are required.

– Về mặt đại số, 1 đường thẳng là 1 phương trình có dạng $a_1x_1 + a_2x_2 = a_0$ với 1 hệ số cố định a_1, a_2, a_0. Hoặc, trong chiều n, ta gọi nó là 1 siêu phẳng & nó có phương trình $\sum_{i=1}^{n} a_i x_i = a_0$. Việc 1 điểm x cụ thể nằm trên 1 trong hai phía của đường thẳng có nghĩa là gì? Ví dụ, $a_1x_1 + a_2x_2 < a_0$ hoặc $a_1x_1 + a_2x_2 > a_0$. Các bất đẳng thức chặt chẽ này có thể được chia tỷ lệ để tăng khoảng cách theo bất kỳ lượng nào. Do đó, có thể đơn giản hóa nhiệm vụ của chúng ta để xác định 1 vectơ a s.t., đối với mọi điểm x' trong lớp A, chúng ta có $\sum_i a_i x'_i \geq a_0 + 1$ & rằng, đối với mọi điểm x'' trong lớp B, chúng ta có $\sum_i a_i x''_i \leq a_0 - 1$. Đưa vào 1 biến dương cho mỗi điểm dữ liệu, giả sử y'_i cho mỗi điểm thuộc lớp A & y''_i cho mỗi điểm thuộc lớp B. Bây giờ, bất đẳng thức $\sum_i a_i x_i \geq a_0 + 1$ có thể được thực thi bằng cách yêu cầu

$$y' \geq a_0 + 1 - \sum_i a_i x'_i,\ y' \geq 0,$$

& tối thiểu hóa y' thành 0. Đại số là đối xứng đối với các điểm thuộc lớp B. Tóm lại, ta được bài toán tối ưu hóa sau:

$$\min \sum_{i \in A} y'_i + \sum_{i \in B} y''_i \text{ với } \begin{cases} y' \geq a_0 + 1 - \sum_i a_i x'_i, \\ y'' \geq \sum_i a_i x''_i - a_0 + 1, \\ y', y'' \geq 0. \end{cases}$$

1 đặc điểm của mô hình này: nếu giá trị mục tiêu tối ưu là 0, chúng ta có 1 siêu phẳng phân tách chính xác tập huấn luyện thành tế bào ác tính & tế bào lành tính. Nhưng nếu giá trị khác không, tức là tập huấn luyện không thể phân tách bằng siêu phẳng & nên cần các kỹ thuật phức tạp hơn.

* 2.5.2. Executable Model. Translate this into executable model seen in Listing 2.7: Identification of Classifying Hyperplane. It accepts 2 sets of data points with any number of features, classified by some expert into classes A & B. After defining potential deviation from hyperplane of sets A & B on lines 4 & 5, define variable that will hold hyperplane on line 6. Note: need this hyperplane later on, to do classification of unknown points. Note also: coefficients could be restricted to be in any interval containing 0. It is simple to scale all coefficients of a plane to have its algebraic expression reside on whatever interval we choose, as long as it includes 0. The constraints at lines 8 & 10 set up offset of each point to hyperplane which objective function will attempt to minimize to 0.

– Biên dịch mô hình này thành mô hình thực thi được thấy trong Liệt kê 2.7: Nhận dạng siêu phẳng phân loại. Nó chấp nhận 2 tập hợp các điểm dữ liệu với bất kỳ số lượng tính năng nào, được 1 số chuyên gia phân loại thành các lớp A & B. Sau khi xác định độ lệch tiềm ẩn khỏi siêu phẳng của các tập hợp A & B trên dòng 4 & 5, hãy xác định biến sẽ giữ siêu phẳng trên dòng 6. Lưu ý: cần siêu phẳng này sau đó để phân loại các điểm chưa biết. Cũng lưu ý: các hệ số có thể bị giới hạn ở bất kỳ khoảng nào chứa 0. Thật đơn giản để chia tỷ lệ tất cả các hệ số của 1 mặt phẳng để biểu thức đại số của nó nằm trên bất kỳ khoảng nào chúng ta chọn, miễn là nó bao gồm 0. Các ràng buộc tại dòng 8 & 10 thiết lập độ lệch của mỗi điểm đến siêu phẳng mà hàm mục tiêu sẽ cố gắng giảm thiểu xuống 0.

Reader might feel a little uncomfortable about this model & here is why, at least partly: This is a model where we do not care about optimal value, but only whether it is 0 or not. Decision variables (& already discussed why this expression is such a misnomer) are not deciding anything. Set of y variables has no real interpretation other than it represents by how much a point violates a linear inequality. & finally, only part of solution we extract, hyperplane, is not used yet. It will only be used later on, in a different program trying to classify a new point as belonging to class A or B. We have moved, with this model, to a higher abstract plane than ever before.

– Người đọc có thể cảm thấy hơi khó chịu về mô hình này & đây là lý do, ít nhất là 1 phần: Đây là mô hình mà chúng ta không quan tâm đến giá trị tối ưu, mà chỉ quan tâm đến việc nó bằng 0 hay không. Các biến quyết định (& đã thảo luận tại sao biểu thức này lại bị gọi sai tên như vậy) không quyết định bất cứ điều gì. Tập hợp y biến không có cách diễn giải thực sự nào khác ngoài việc nó biểu thị mức độ vi phạm bất đẳng thức tuyến tính của 1 điểm. & Cuối cùng, chỉ 1 phần của nghiệm mà chúng ta trích xuất, siêu phẳng, vẫn chưa được sử dụng. Nó sẽ chỉ được sử dụng sau này, trong 1 chương trình khác cố gắng phân loại 1 điểm mới thành lớp A hoặc B. Với mô hình này, chúng ta đã chuyển sang 1 mặt phẳng trừu tượng cao hơn bao giờ hết.

· 2.5.2.1. Variations. There are at least 3 directions we can go from this model.

1. 1st: add constraints to increase quality of returned hyperplane. E.g., we could require: it not only separates 2 sets, but that it is, in some sense, as far from 1 set as from the other. If training set is well-chosen, this will ensure that we minimize erroneous classifications later on. This is known as maximizing margin & will tackle this problem in a later chap.
2. 2nd direction to pursue is what to do when optimal value is not 0, i.e., when 2 sets are not separable by a hyperplane. They may be separable by a nonlinear curve. This question is complex & multiple approaches have been tried, but most rely on knowing something additional about data. Will not consider it.
3. Final improvement would be to consider classification into multiple classes, considered in a later chap.

– Có ít nhất 3 hướng chúng ta có thể đi từ mô hình này.

1. 1st: thêm các ràng buộc để tăng chất lượng siêu phẳng trả về. Ví dụ, chúng ta có thể yêu cầu: nó không chỉ tách 2 tập hợp, mà theo 1 nghĩa nào đó, nó phải cách xa tập hợp này cũng như tập hợp kia. Nếu tập huấn luyện được chọn tốt, điều này sẽ đảm bảo chúng ta giảm thiểu các phân loại sai sau này. Điều này được gọi là tối đa hóa biên độ & sẽ giải quyết vấn đề này trong 1 chương sau.
2. 2th direction cần theo đuổi là phải làm gì khi giá trị tối ưu khác 0, tức là khi 2 tập hợp không thể tách rời nhau bằng 1 siêu phẳng. Chúng có thể tách rời nhau bằng 1 đường cong phi tuyến tính. Câu hỏi này khá phức tạp & nhiều phương pháp đã được thử nghiệm, nhưng hầu hết đều dựa vào kiến thức bổ sung về dữ liệu. Tôi sẽ không xem xét điều này.
3. Cải tiến cuối cùng sẽ là xem xét phân loại thành nhiều lớp, sẽ được xem xét trong 1 chương sau.

• 3. Hidden Linear Continuous Models. In this chap, do violence to some problems to reveal their inner structure. Focus is on problems which, at 1st glance, may not seem to be of continuous linear variety yet can be marshalled into that form with a handful of creative alternations. Key: assure a 1-1 correspondence between original & altered problems so that we can retrieve a solution to original from a solution to alternation.

– Trong chương này, hãy thử nghiệm 1 số bài toán để khám phá cấu trúc bên trong của chúng. Trọng tâm là các bài toán mà thoạt nhìn có vẻ không phải là dạng tuyến tính liên tục, nhưng có thể được sắp xếp thành dạng đó bằng 1 số phép biến đổi sáng tạo. Điểm mấu chốt: đảm bảo sự tương ứng 1-1 giữa bài toán gốc & đã biến đổi để ta có thể tìm ra lời giải cho bài toán gốc từ lời giải cho phép biến đổi.

Main reason for massaging problems in this way: continuous linear solvers have become so fast that they can handle models with hundreds of thousands of variables & constraints. Therefore, if a problem can be modeled in that manner, there is little practical limit on instance size that can be solved. As see later, this is not case with more complex models. In fact, can write models with a few dozen variables that no current solver in a reasonable time.

– Lý do chính cho việc xử lý các bài toán theo cách này: các bộ giải tuyến tính liên tục đã trở nên quá nhanh đến mức chúng có thể xử lý các mô hình với hàng trăm nghìn biến & ràng buộc. Do đó, nếu 1 bài toán có thể được mô hình hóa theo cách đó, sẽ có rất ít giới hạn thực tế về kích thước thực thể có thể được giải quyết. Như sẽ thấy sau, điều này không đúng với các mô hình phức tạp hơn. Trên thực tế, có thể viết các mô hình với vài chục biến mà không có bộ giải hiện tại nào có thể làm được trong 1 khoảng thời gian hợp lý.

Main obstacles encountered in this chap are nonlinearities of 1 kind or another, but with advantageous restriction: functions be considered convex. A convex function [All research mathematicians agree on labels "convex" & its opposite "concave", but textbook authors for high schools in US, ignoring thousands of papers, journals, & research monographs, insist on "concave up" & "concave down".] is one that sits "above" any valid linear approximation to it. In 1D, algebraically, f is convex at point x_0 if $f(x_0 + h) \geq f(x_0) + f'(x_0)h$. Geometrically, it looks like Fig. 3.1: Prototypical example of a convex function & a linear approximation, with a 1st-order approximation of $f(x) = x^2$ at $x_0 = 4$. Convexity will be Trojan horse used to beat nonlinearity into submission.

– Những trở ngại chính gặp phải trong chương này là các tính phi tuyến tính thuộc loại này hay loại khác, nhưng với 1 hạn chế có lợi: các hàm được coi là lồi. 1 hàm lồi [Tất cả các nhà toán học nghiên cứu đều đồng ý về nhãn "lồi" & "lõm" đối lập của nó, nhưng các tác giả sách giáo khoa cho học sinh trung học ở Hoa Kỳ, bỏ qua hàng ngàn bài báo, tạp chí, & chuyên khảo nghiên cứu, lại khăng khăng dùng "lõm lên" & "lõm xuống".] là 1 hàm nằm "trên" bất kỳ phép xấp xỉ tuyến tính hợp lệ nào đối với nó. Trong không gian 1 chiều, về mặt đại số, f là lồi tại điểm x_0 nếu $f(x_0 + h) \geq f(x_0) + f'(x_0)h$. Về mặt hình học, nó trông giống như Hình 3.1: Ví dụ nguyên mẫu của hàm lồi & xấp xỉ tuyến tính, với xấp xỉ bậc 1 của $f(x) = x^2$ tại $x_0 = 4$. Độ lồi sẽ là con ngựa thành Troy được sử dụng để đánh bại tính phi tuyến tính.

◦ 3.1. Piecewise Linear. Consider here broken-up linear functions. In traditional parlance, they are *piecewise linear*. As such, linear programming solvers we have used up to now (GLPK, GLOP, CLP) cannot handle them directly, but a little coding

on our part will morph them into a standard form that all solvers can handle. This is 1 of good reasons to code models in Python instead of specialized modeling languages.

– Hãy xem xét các hàm tuyến tính bị chia nhỏ ở đây. Theo cách nói truyền thống, chúng là *tuyến tính từng phần*. Do đó, các trình giải quy hoạch tuyến tính mà chúng ta đã sử dụng cho đến nay (GLPK, GLOP, CLP) không thể xử lý chúng trực tiếp, nhưng 1 chút mã hóa từ phía chúng ta sẽ biến chúng thành 1 dạng chuẩn mà tất cả các trình giải đều có thể xử lý. Đây là 1 trong những lý do chính đáng để viết mã mô hình bằng Python thay vì các ngôn ngữ mô hình hóa chuyên biệt.

p. 65+++

* 6.3.1. Constructing a Model.
* 6.3.2. Variations.

- 6.4. Job Shop Scheduling. A rather difficult problem to solve: consider set of jobs J to be performed on set of machines M. Each job requires some time on each of machines to perform a specific task & has an order in which tasks must be performed. Can think of building wooden toys. Wood needs to be cut to shape, then sanded, then primed, then painted, then covered in lacquer. Each of these tasks is accomplished by a different machine & requires a certain duration, which depends on toy. Word "machine" here is used in a rather wide sense. It could indicate a highly trained worker at his station. It could be a robot. Idea is same. Moreover, not all jobs need all machines. Some may require only a subset of machines. Overall goal: schedule time on each machine to do all tasks while minimizing overall time used. Small example we will use to illustrate this model is given in Table 6.10: xample of Job Shop Scheduling (for Each Job, Machine & Duration of Tasks).
  – 1 bài toán khá khó giải: xét tập hợp các công việc J cần được thực hiện trên tập hợp các máy M. Mỗi công việc cần 1 khoảng thời gian nhất định trên mỗi máy để thực hiện 1 nhiệm vụ cụ thể & có thứ tự các công việc phải được thực hiện. Hãy nghĩ đến việc làm đồ chơi bằng gỗ. Gỗ cần được cắt theo hình dạng, sau đó chà nhám, sơn lót, sơn phủ, rồi phủ vecni. Mỗi công việc này được thực hiện bởi 1 máy khác nhau & cần 1 khoảng thời gian nhất định, tùy thuộc vào từng loại đồ chơi. Từ "máy" ở đây được sử dụng theo nghĩa khá rộng. Nó có thể chỉ 1 công nhân được đào tạo bài bản tại vị trí của anh ta. Nó cũng có thể là 1 robot. Ý tưởng là như nhau. Hơn nữa, không phải tất cả các công việc đều cần tất cả các máy. 1 số có thể chỉ cần 1 tập hợp con các máy. Mục tiêu chung: lên lịch thời gian cho mỗi máy để thực hiện tất cả các nhiệm vụ trong khi giảm thiểu tổng thời gian sử dụng. Ví dụ nhỏ chúng ta sẽ sử dụng để minh họa mô hình này được đưa ra trong Bảng 6.10: Ví dụ về Lập lịch Xưởng Công việc (cho Mỗi Công việc, Máy & Thời lượng Nhiệm vụ).

  * 6.4.1. Constructing a Model. Since we are given order of tasks for each job, it is not part of decision. Time at which a particular task is started for a given job on a given machine is what we seek. Therefore use a decision variable indicating starting time $x_{ij} \in [0, \infty)$, $\forall i \in J, \forall k \in M$. We are not making assumption that duration are given as integers so that decision variable is continuous.
    – Vì chúng ta được cung cấp thứ tự các tác vụ cho mỗi công việc, nên việc này không phải là 1 phần của quyết định. Thời điểm bắt đầu 1 tác vụ cụ thể cho 1 công việc nhất định trên 1 máy nhất định là điều chúng ta cần. Do đó, hãy sử dụng 1 biến quyết định biểu thị thời điểm bắt đầu $x_{ij} \in [0, \infty)$, $\forall i \in J, \forall k \in M$. Chúng ta không giả định rằng thời lượng được cho dưới dạng số nguyên để biến quyết định là liên tục.
    1st type of constraints puts a lower bound on starting times. Consider, say, job 7 must use machine 4 for 3 hours before it goes to machine 6, then starting time is $x_{74} + 3 \leq x_{76}$. In general, assuming p_i is vector of order of machines of job i (a permutation of $0, 1, 2, \ldots, M-1$) with duration vector d_i, we are led to

$$x_{ip_{ik}} + d_{ik} \leq x_{ip_{ik+1}}, \ \forall i \in J, \ \forall k \in [0, \ldots, |M| - 2].$$

    Difficulty in this problem stems from enforcing that, at any given time, there is at most 1 job per machine. Acceptable for a machine to be idle, though we will try to minimize idleness. Howe can we enforce this constraint?
    – Loại ràng buộc thứ nhất đặt ra giới hạn dưới cho thời gian bắt đầu. Giả sử, công việc 7 phải sử dụng máy 4 trong 3 giờ trước khi đến máy 6, khi đó thời gian bắt đầu là $x_{74} + 3 \leq x_{76}$. Nhìn chung, giả sử p_i là vectơ bậc máy của công việc i (hoán vị của $0, 1, 2, \ldots, M-1$) với vectơ thời gian d_i, ta được

$$x_{ip_{ik}} + d_{ik} \leq x_{ip_{ik+1}}, \ \forall i \in J, \ \forall k \in [0, \ldots, |M| - 2].$$

    Khó khăn trong bài toán này bắt nguồn từ việc bắt buộc rằng, tại bất kỳ thời điểm nào, mỗi máy chỉ được phép thực hiện tối đa 1 công việc. Có thể chấp nhận máy ở trạng thái nhàn rỗi, mặc dù chúng ta sẽ cố gắng giảm thiểu tình trạng nhàn rỗi. Làm thế nào chúng ta có thể thực thi ràng buộc này?
    1 way: introduce an additional variable that will indicate relative order of jobs on a given machine. E.g., $z_{ijk} \in \{0, 1\}$, $\forall i, j \in J, \forall k \in M$, with interpretation that $z_{ijk} = 1$ iff job i precedes job j on machine k. Note, as either job i precedes job j or vice-versa, $z_{ijk} = 1 \Leftrightarrow z_{jik} = 0$. With this variable we can try to enforce our difficult condition. Consider again a simple example. Say that job 7 needs machine 2 for 3 hours & job 5 needs it for 4 hours. Then either $x_{72} + 3 \leq x_{52}$, indicating job 5 cannot start until job 7 has ended on machine 2, or else $x_{52} + 4 \leq x_{72}$ indicating reverse temporal condition. This is a disjunction; we need 1 or the other of constraints to hold.
    – Cách 1: thêm 1 biến bổ sung để chỉ ra thứ tự tương đối của các công việc trên 1 máy nhất định. Ví dụ: $z_{ijk} \in \{0, 1\}$, $\forall i, j \in J, \forall k \in M$, với cách diễn giải là $z_{ijk} = 1$ nếu & chỉ nếu công việc i xảy ra trước công việc j trên máy k. Lưu ý, vì công việc i xảy ra trước công việc j hoặc ngược lại, $z_{ijk} = 1 \Leftrightarrow z_{jik} = 0$. Với biến này, ta có thể thử áp dụng điều kiện khó khăn của mình. Hãy xem xét lại 1 ví dụ đơn giản. Giả sử công việc 7 cần máy 2 trong 3 giờ & công việc 5 cần nó trong 4 giờ. Khi đó, hoặc $x_{72} + 3 \leq x_{52}$, biểu thị công việc 5 không thể bắt đầu cho đến khi công việc 7 kết thúc trên máy 2, hoặc $x_{52} + 4 \leq x_{72}$ biểu thị điều kiện thời gian ngược lại. Đây là 1 phép phân ly; chúng ta cần 1 hoặc 2 ràng buộc phải thỏa mãn.
    Using variables z_{ijk} this condition can be enforced by considering, for all jobs i, j & all machines k,

$$\begin{aligned} z_{ijk} &= 1 - z_{jik}, \\ x_{ip_{ik}} + d_{ik} - M z_{ijp_{ik}} &\leq x_{jp_{ik}}, \\ x_{jp_{jk}} + d_{jk} - M z_{jip_{jk}} &\leq x_{ip_{jk}}, \end{aligned}$$

    for some large enough value M. Note only 1 of 2 z_{ijk} or z_{jik} will be 1. Hence exactly 1 of 2 inequalities will constraining. The other will be vacuously satisfied.

– Sử dụng các biến z_{ijk}, điều kiện này có thể được thực thi bằng cách xem xét, đối với mọi công việc i, j & mọi máy k,

$$z_{ijk} = 1 - z_{jik},$$
$$x_{ip_{ik}} + d_{ik} - M z_{ijp_{ik}} \leq x_{jp_{ik}},$$
$$x_{jp_{jk}} + d_{jk} - M z_{jip_{jk}} \leq x_{ip_{jk}},$$

với 1 giá trị M đủ lớn. Lưu ý rằng chỉ 1 trong 2 z_{ijk} hoặc z_{jik} sẽ bằng 1. Do đó, đúng 1 trong 2 bất đẳng thức sẽ ràng buộc. Bất đẳng thức còn lại sẽ được thỏa mãn 1 cách trống rỗng.
All that is left: deal with objective function. We want a schedule that will complete all jobs in least amount of time. We could add a bound to end of each job & minimize that bound. E.g.,

$$x_{ip_{ik}} + d_{ip_{ik}} \leq T, \ \forall i \in J, \ \forall k \in [0, \ldots, |M| - 1],$$

with objective $\min T$ where T will be completion time of last task on last machine.
– Tất cả những gì còn lại: xử lý hàm mục tiêu. Chúng ta muốn 1 lịch trình hoàn thành tất cả các công việc trong thời gian ngắn nhất. Chúng ta có thể thêm 1 giới hạn vào cuối mỗi công việc & tối thiểu hóa giới hạn đó. Ví dụ:

$$x_{ip_{ik}} + d_{ip_{ik}} \leq T, \forall i \in J, \forall k \in [0, \ldots, |M| - 1],$$

với mục tiêu $\min T$ trong đó T sẽ là thời gian hoàn thành tác vụ cuối cùng trên máy cuối cùng.

· 6.4.1.1. Executable Model. Executable code is seen in Listing 6.4. Job Shop Scheduling Model. It assumes: input is a list of tuples exactly as in Table 6.10, indicating for each job order of machines needed with duration of task on each machine.
– Mã thực thi được hiển thị trong Liệt kê 6.4. Mô hình Lập lịch Xưởng Gia công. Mã này giả định: đầu vào là 1 danh sách các bộ dữ liệu chính xác như trong Bảng 6.10, cho biết số lượng máy cần thiết cho mỗi lệnh gia công cùng với thời lượng thực hiện công việc trên mỗi máy.
At line 4, compute a value larger than last possible end time by adding all durations of all tasks as if we had only 1 machine. This will be used to limit domain of decision variables as well as bounding some constraints. Lines 6–7 extract from input tuples a vector of permutation of machines & a vector of durations. This will allow us to write code as closely as possible to abstract mathematical description we used.
– Ở dòng 4, hãy tính 1 giá trị lớn hơn thời gian kết thúc khả thi cuối cùng bằng cách cộng tất cả thời lượng của tất cả các tác vụ như thể chúng ta chỉ có 1 máy. Điều này sẽ được sử dụng để giới hạn miền biến quyết định cũng như giới hạn 1 số ràng buộc. Dòng 6-7 trích xuất từ các bộ dữ liệu đầu vào 1 vectơ hoán vị của các máy & 1 vectơ thời lượng. Điều này sẽ cho phép chúng ta viết mã gần nhất có thể với mô tả toán học trừu tượng mà chúng ta đã sử dụng.
Line 13 sets T as upper bound on all end times of all tasks. Then minimize this T to obtain shortest schedule. Line 22 is translation of (6.4.1), enforcing: for each job, tasks are done in order prescribed by input tuples.
– Dòng 13 đặt T làm giới hạn trên cho tất cả thời gian kết thúc của tất cả các tác vụ. Sau đó, tối thiểu hóa T này để có được lịch trình ngắn nhất. Dòng 22 là phép dịch của (6.4.1), thực thi: đối với mỗi công việc, các tác vụ được thực hiện theo thứ tự được quy định bởi các bộ dữ liệu đầu vào.
Finally, `if` block starting at line 15 is where complex disjunction occurs. It enforces that, for every pair of tasks, one strictly precedes the other. Possible to reduce number of variables by half, but a good solver will do that automatically during a pre-processing phase.
– Cuối cùng, khối `if` bắt đầu từ dòng 15 là nơi xảy ra phân tách phức hợp. Khối này đảm bảo rằng, đối với mỗi cặp tác vụ, tác vụ này phải được thực hiện trước tác vụ kia. Có thể giảm số lượng biến xuống 1 nửa, nhưng 1 trình giải tốt sẽ tự động thực hiện việc này trong giai đoạn tiền xử lý.
Result of our simple example is shown in Table 6.11: Optimal Solution (S: Start Time, M: Machine, D: Duration) & Fig. 6.1: Graphical representation of schedule.
– Kết quả của ví dụ đơn giản của chúng tôi được hiển thị trong Bảng 6.11: Giải pháp tối ưu (S: Thời gian bắt đầu, M: Máy, D: Thời lượng) & Hình 6.1: Biểu diễn đồ họa của lịch trình.

- 7. Advanced Techniques. In this chap, cover some of tricks optimizers have developed over years to twist models & cajole solvers into providing solutions for problems that do not easily fit Procrustean rules of mathematical optimization.

– Trong chương này, chúng ta sẽ tìm hiểu 1 số thủ thuật mà các nhà tối ưu hóa đã phát triển trong nhiều năm qua để biến đổi các mô hình & dụ dỗ người giải đưa ra giải pháp cho các vấn đề không dễ dàng phù hợp với các quy tắc Procrustean về tối ưu hóa toán học.

Some of these techniques involve iteratively solving a sequence of partial models, converging to solution we seek. Some involve create use of layers upon layers of decision variables, each abstracting an additional level of details. Some involve inspection of an invalid model to gain insight into an improved one. All in all, it's a mixed bag of jewels with little in common but goal of solving more complex problems.

– 1 số kỹ thuật này liên quan đến việc giải quyết lặp đi lặp lại 1 chuỗi các mô hình từng phần, hội tụ về giải pháp chúng ta tìm kiếm. 1 số liên quan đến việc sử dụng các lớp biến quyết định chồng lên nhau, mỗi lớp trừu tượng hóa 1 cấp độ chi tiết bổ sung. 1 số liên quan đến việc kiểm tra 1 mô hình không hợp lệ để hiểu sâu hơn về 1 mô hình được cải thiện. Nhìn chung, đây là 1 tập hợp các bài toán phức tạp với rất ít điểm chung nhưng mục tiêu là giải quyết các vấn đề phức tạp hơn.

End with a series of puzzles that are rarely considered by classical optimizers but are bread & butter of constraint programmers. To solve them, build a small library of functions that proves invaluable in succinctly expressing these models, proving, I trust, that with right tools & right language, expressive power of constraint programming can be carried over to integer programming.

– Kết thúc bằng 1 loạt câu đố hiếm khi được các nhà tối ưu hóa cổ điển xem xét nhưng lại là công cụ đắc lực của các lập trình viên ràng buộc. Để giải quyết chúng, hãy xây dựng 1 thư viện hàm nhỏ, chứng minh được giá trị của nó trong việc diễn đạt ngắn gọn các mô hình này, & tôi tin rằng, với đúng công cụ & đúng ngôn ngữ, sức mạnh biểu đạt của lập trình ràng buộc có thể được chuyển giao sang lập trình số nguyên.

◦ 7.1. Cutting Stock. Cutting stock problem originated in paper & textile industries. A typical mill produces paper in very large product rolls (large in length & in width), which are then cut to width required by customers. Call later *consumer rolls*. E.g., tabloid printers want their paper in rolls 17 inches wide, while broadsheet printers want twice that width. These consumer rolls are cut to page size after printing. Cutting problem, for paper producer, is to cut large-width rolls to satisfy customer demands while minimizing waste.

– Vấn đề cắt giấy tồn kho bắt nguồn từ ngành công nghiệp giấy & dệt may. 1 nhà máy giấy thông thường sản xuất giấy thành những cuộn sản phẩm rất lớn (dài & rộng), sau đó được cắt theo chiều rộng yêu cầu của khách hàng. Gọi sau *cuộn tiêu dùng*. Ví dụ, máy in báo khổ nhỏ muốn giấy của họ ở dạng cuộn rộng 17 inch, trong khi máy in khổ lớn muốn chiều rộng gấp đôi. Những cuộn tiêu dùng này được cắt theo kích thước trang sau khi in. Vấn đề cắt giấy, đối với nhà sản xuất giấy, là cắt những cuộn khổ lớn để đáp ứng nhu cầu của khách hàng đồng thời giảm thiểu lãng phí.

Table 7.1: Example of Cutting Stock Problem is a random instance of orders we will solve to illustrate model. 1st column is number of consumer rolls in order & 2nd column is required width that has been pre-processed to be measured in percentage of product roll width.

– Bảng 7.1: Ví dụ về bài toán cắt hàng tồn kho là 1 trường hợp ngẫu nhiên của các đơn hàng mà chúng ta sẽ giải quyết để minh họa cho mô hình. Cột thứ nhất là số lượng cuộn tiêu dùng theo đơn hàng & Cột thứ hai là chiều rộng bắt buộc đã được xử lý trước để đo theo phần trăm chiều rộng cuộn sản phẩm.

Want to minimize paper waste, which really means minimizing total number of product rolls used. But, in order to do cutting, will need more information, namely given any product roll, how is it to be cut? Say we have a number of orders for 21% & for 36%, do we cut at 21%, 42%, 63%, & 99% with a waste of 1% or at 36%, 72%, & 93% for a waste of 7%? It may be that we need to cut half our rolls with 1st pattern & half with 2nd to satisfy all demands. Which patterns to cut & how many rolls of a given pattern are key problems. See Fig. 7.1: Which pattern do we use?.

– Muốn giảm thiểu lãng phí giấy, nghĩa là giảm thiểu tổng số cuộn sản phẩm được sử dụng. Tuy nhiên, để thực hiện việc cắt, cần thêm thông tin, cụ thể là với bất kỳ cuộn sản phẩm nào, cần cắt như thế nào? Giả sử chúng ta có 1 số đơn hàng với 21% & cho 36%, chúng ta sẽ cắt ở mức 21%, 42%, 63%, & 99% với mức lãng phí là 1% hay ở mức 36%, 72%, & 93% với mức lãng phí là 7%? Có thể chúng ta cần cắt 1 nửa số cuộn với mẫu 1 & 1 nửa với mẫu 2 để đáp ứng tất cả các nhu cầu. Những mẫu nào cần cắt & bao nhiêu cuộn của 1 mẫu nhất định là những vấn đề chính. Xem Hình 7.1: Chúng ta sử dụng mẫu nào?.

∗ 7.1.1. Constructing a Model. Originally problem was attacked by specifying various cutting patterns, sometimes statistically, sometimes dynamically. This was more or less forced by technology of time. Things have changed; both processors & solvers are considerably faster. Moreover, a good modeler should try simplest approach 1st & make it more complex iff it fails. With this in mind, we will leave to solver problem of deciding on patterns.

– Ban đầu, bài toán được giải quyết bằng cách xác định các mẫu cắt khác nhau, đôi khi theo thống kê, đôi khi theo động. Điều này ít nhiều đã được áp dụng bởi công nghệ thời đại. Mọi thứ đã thay đổi; cả bộ xử lý & trình giải đều nhanh hơn đáng kể. Hơn nữa, 1 nhà mô hình giỏi nên thử phương pháp đơn giản nhất trước & làm cho nó phức tạp hơn nếu thất bại. Với suy nghĩ này, chúng ta sẽ để bài toán quyết định mẫu cho người giải.

· 7.1.1.1. Decision Variables. If leave pattern decision to solver, problem becomes, given a roll, where do we cut? But this is over-specifying. Pattern 21, 42, 63, 99 is not distinguishable from pattern 21, 57, 78, 99. They satisfy exactly same customer demands: 3 21$ & 1 36% width. & we know: having multiple indistinguishable solutions is very bad for a solver. Therefore, what we should ask is "Given a roll, how many cuts of width w do we make?"

– Nếu để người giải tự quyết định mẫu, bài toán sẽ trở thành, với 1 lần tung xúc xắc, chúng ta sẽ cắt ở đâu? Nhưng điều này là quá cụ thể. Mẫu 21, 42, 63, 99 không thể phân biệt được với mẫu 21, 57, 78, 99. Chúng đáp ứng chính xác cùng 1 yêu cầu của khách hàng: chiều rộng 3 21$ & 1 36%. & chúng ta biết: việc có nhiều giải pháp không thể phân biệt được là rất tệ cho người giải. Do đó, điều chúng ta nên hỏi là "Với 1 lần tung xúc xắc, chúng ta sẽ thực hiện bao nhiêu lần cắt có chiều rộng w?"

With this in mind, assuming that we have N orders & at most K rolls, a reasonable decision variable is

$$x_{ij} \in \mathbb{N},\ \forall i \in N,\ \forall j \in K,$$

where $x_{2,5} = 7$, e.g., will mean: we cut roll 5 to satisfy 7 customer rolls of width specified by order 2. Ordering of cuts is irrelevant but we might very well post-process solution to produce cutting patterns to be used.

– Với suy nghĩ này, giả sử chúng ta có N đơn hàng & tối đa K lần tung, 1 biến quyết định hợp lý là

$$x_{ij} \in \mathbb{N},\ \forall i \in N,\ \forall j \in K,$$

trong đó $x_{2,5} = 7$, ví dụ, sẽ có nghĩa là: chúng ta cắt cuộn 5 để thỏa mãn 7 cuộn khách hàng có chiều rộng được chỉ định bởi thứ tự 2. Thứ tự cắt không liên quan nhưng chúng ta có thể xử lý hậu kỳ giải pháp để tạo ra các mẫu cắt sẽ sử dụng.

& since we do not know ahead of time how many rolls we will need, there has to be a corresponding variable to indicate roll use,

$$y_j \in \{0, 1\},\ \forall j \in K,$$

where $y_5 = 1$ means roll 5 (outt of a possible K) is used. This is same trick used in facility location problem to decide which facility to open. Maximum number of rolls K does not have to be precisely determined; any upper bound will do at this point.

– & Vì chúng ta không biết trước cần bao nhiêu lần tung xúc xắc, nên phải có 1 biến tương ứng để chỉ ra số lần tung xúc xắc cần dùng,

$$y_j \in \{0, 1\},\ \forall j \in K,$$

trong đó $y_5 = 1$ nghĩa là lần tung xúc xắc thứ 5 (trong tổng số K có thể) được sử dụng. Đây cũng là mẹo được sử dụng trong bài toán xác định vị trí cơ sở để quyết định cơ sở nào cần mở. Số lần tung xúc xắc tối đa K không cần phải xác định chính xác; bất kỳ giới hạn trên nào cũng được.

· 7.1.1.2. Objective. Objective: minimize number of rolls. Could minimize sum of all y_j, but that leaves open possibility of a solution that says "Use rolls 1 & 3, but not 2." To avoid such annoyance, use a small trick that will make each new roll more costly than the last: $\min \sum_{j \in K} jy_j$. But now our objective function value at optimality will not represent number of rolls used, so introduce an auxiliary variable, say t, & add $t = \sum_j y_j$.

– Mục tiêu: giảm thiểu số lần tung. Có thể giảm thiểu tổng của tất cả y_j, nhưng điều đó vẫn còn bỏ ngỏ khả năng có 1 giải pháp là "Sử dụng lần tung 1 & 3, nhưng không phải lần tung 2." Để tránh sự phiền toái này, hãy sử dụng 1 mẹo nhỏ khiến mỗi lần tung mới tốn kém hơn lần trước: $\min \sum_{j \in K} jy_j$. Nhưng bây giờ, giá trị hàm mục tiêu của chúng ta ở trạng thái tối ưu sẽ không biểu thị số lần tung đã sử dụng, vì vậy hãy thêm 1 biến phụ, chẳng hạn t, & cộng $t = \sum_j y_j$.

· 7.1.1.3. Constraints. Have 2 types of constraints. 1st: ensure customer demands are satisfied. So, across all rolls used, must verify we cut enough rolls of required order quantity, say b_i,

$$\sum_{j \in K} x_{i,j} \geq b_i,\ \forall i \in N.$$

2nd type of constraints: make sure consumer rolls we cut off product roll do not exceed width of that large roll, or, assuming that order i has width w_i, a percentage,

$$\sum_{i \in N} w_i x_{ij} \leq 100,\ \forall j \in K.$$

But that is incomplete since we need to connect x, y variables. No x_{ij} should be positive if corresponding y_j is 0. Could introduce a large number of constraints or, realizing that we have encountered this situation before, simply modify last constraint to read

$$\sum_{i \in N} w_i x_{ij} \leq 100 y_j,\ \forall j \in K.$$

– Có 2 loại ràng buộc. Thứ nhất: đảm bảo đáp ứng nhu cầu của khách hàng. Vì vậy, trên tất cả các cuộn được sử dụng, phải xác minh rằng chúng ta đã cắt đủ số cuộn với số lượng yêu cầu, giả sử là b_i,

$$\sum_{j \in K} x_{i,j} \geq b_i,\ \forall i \in N.$$

Loại ràng buộc thứ hai: đảm bảo các cuộn tiêu dùng mà chúng ta cắt ra khỏi cuộn sản phẩm không vượt quá chiều rộng của cuộn lớn đó, hoặc, giả sử rằng đơn hàng i có chiều rộng w_i, 1 tỷ lệ phần trăm,

$$\sum_{i \in N} w_i x_{ij} \leq 100,\ \forall j \in K.$$

Nhưng điều đó chưa đầy đủ vì chúng ta cần kết nối các biến x, y. Không có x_{ij} nào dương nếu y_j tương ứng bằng 0. Có thể đưa ra 1 số lượng lớn các ràng buộc hoặc, nhận ra rằng chúng ta đã gặp phải tình huống này trước đây, chỉ cần sửa đổi ràng buộc cuối cùng thành đọc

$$\sum_{i \in N} w_i x_{ij} \leq 100 y_j,\ \forall j \in K.$$

· 7.1.1.4. Executable Model. Translate this into an executable model, as shown in Listing 7.1: Cutting Stock Model with Pattern Search. It accepts a matrix D, exactly as in Table 7.1.

– Chuyển đổi mô hình này thành 1 mô hình thực thi, như được hiển thị trong Liệt kê 7.1: Mô hình Cắt Cổ phiếu với Tìm kiếm Mẫu. Nó chấp nhận 1 ma trận D, chính xác như trong Bảng 7.1.

At line 3 we call a routine bounds to find lower & upper bounds on numbers of rolls we will require & on number of rolls of each order that can fit in a roll. Upper bound on number of rolls used on line 4 to create as many y as we could possibly need (& also on subsequent constraints related to each roll). Bound of number of rolls of each type is used at next line to establish range for each x: on any given roll, we can have 0 up to number of elements that can fit or number required by customer, hence min expression.

– Ở dòng 3, chúng ta gọi 1 hàm biên để tìm giới hạn dưới & giới hạn trên của số lần tung cần thiết & giới hạn trên của số lần tung của mỗi thứ tự có thể vừa với 1 lần tung. Giới hạn trên của số lần tung được sử dụng ở dòng 4 để tạo ra càng nhiều y càng tốt (& cũng liên quan đến các ràng buộc tiếp theo liên quan đến mỗi lần tung). Giới hạn số lần tung

của mỗi loại được sử dụng ở dòng tiếp theo để thiết lập phạm vi cho mỗi x: với bất kỳ lần tung nào, chúng ta có thể có từ 0 đến số phần tử có thể vừa hoặc số lượng mà khách hàng yêu cầu, do đó biểu thức min.
Line 10 ensures we satisfy every customer demand by summing over all rolls elements of a given order. Could also have used an inequality here, namely $\geq$. Idea: may cut more than customer demanded & then store these rolls in inventory, awaiting next order. This sometimes makes sense, but is often better left out of model. Once we have a solution that exactly satisfies customer demand, someone with an understanding of situation & in possession of planning tools can decide how to best cut leftover of rolls. In either case, this will not change total number of rolls used.
– Dòng 10 đảm bảo chúng ta đáp ứng mọi nhu cầu của khách hàng bằng cách cộng tổng tất cả các cuộn giấy của 1 đơn hàng nhất định. Cũng có thể sử dụng 1 bất đẳng thức ở đây, cụ thể là $\geq$. Ý tưởng: có thể cắt nhiều hơn nhu cầu của khách hàng & sau đó lưu trữ các cuộn giấy này trong kho, chờ đơn hàng tiếp theo. Điều này đôi khi hợp lý, nhưng thường tốt hơn là không đưa vào mô hình. Khi chúng ta có 1 giải pháp đáp ứng chính xác nhu cầu của khách hàng, 1 người hiểu rõ tình hình & có công cụ lập kế hoạch có thể quyết định cách cắt giảm tốt nhất số cuộn giấy còn lại. Trong cả hai trường hợp, điều này sẽ không làm thay đổi tổng số cuộn giấy đã sử dụng.
Line 12 ensures: consumer rolls cut off a roll do not add up to $> 100\%$ of roll. & next line, not a constraint, simply, computes waste of each roll to help return a meaningful solution.
– Dòng 12 đảm bảo: các cuộn của người tiêu dùng bị cắt khỏi 1 cuộn không cộng lại thành $> 100\%$ cuộn. & dòng tiếp theo, không phải là ràng buộc, chỉ đơn giản là tính toán lượng lãng phí của mỗi cuộn để giúp trả về 1 giải pháp có ý nghĩa.
Loop starting at 14 breaks symmetry of multiple solutions that are equivalent for our purposes: any permutation of rolls. These permutations, & there are $K!$ of them, cause most solvers to spend an exorbitant time solving. With this constraint, tell solver to prefer those permutations with more cuts in roll j than in roll $j + 1$. Reader is encouraged to solve a medium-sized problem with & without this symmetry-breaking constraint. I have seen problems take 48 hours to solve without constraint & 48 minutes with. Of course, for problems that are solved in seconds, constraint will not help; it may even hinder. But who cares if a cutting stock instance solvers in 2 or in 3 seconds? Care much more about difference between 2 mins & 3 hours, which is what this constraint is meant to address.
– Vòng lặp bắt đầu từ 14 phá vỡ tính đối xứng của nhiều giải pháp tương đương cho mục đích của chúng ta: bất kỳ hoán vị nào của các lần tung. Những hoán vị này, & có $K!$ trong số chúng, khiến hầu hết các trình giải tốn rất nhiều thời gian để giải. Với ràng buộc này, hãy yêu cầu trình giải ưu tiên các hoán vị có nhiều lần cắt hơn trong lần tung j so với lần tung $j + 1$. Người đọc được khuyến khích giải 1 bài toán có kích thước trung bình với & mà không có ràng buộc phá vỡ tính đối xứng này. Tôi đã thấy các bài toán mất 48 giờ để giải mà không có ràng buộc & 48 phút có. Tất nhiên, đối với các bài toán được giải trong vài giây, ràng buộc sẽ không giúp ích gì; nó thậm chí có thể gây cản trở. Nhưng ai quan tâm nếu 1 trường hợp cắt giảm cổ phiếu giải quyết trong 2 hay trong 3 giây? Quan tâm nhiều hơn đến sự khác biệt giữa 2 phút & 3 giờ, đó là mục tiêu mà ràng buộc này nhằm giải quyết.
Could use, for objective function, simple sum of rolls used variable y & pre-process unused rolls. But we used a little trick to make each additional roll more "expensive" by including an ordinal factor. This guarantees that if, say number of rolls is estimated to be between 11 & 14 & end up using 12, they will be the 1st 12. There will be no "holes".
– Có thể sử dụng, cho hàm mục tiêu, tổng đơn giản các lần tung đã dùng biến y & xử lý trước các lần tung chưa dùng. Tuy nhiên, chúng tôi đã sử dụng 1 mẹo nhỏ để làm cho mỗi lần tung bổ sung "đắt hơn" bằng cách thêm vào 1 hệ số thứ tự. Điều này đảm bảo rằng nếu, giả sử số lần tung được ước tính nằm trong khoảng từ 11 & 14 & cuối cùng sử dụng 12, thì chúng sẽ là 12 lần đầu tiên. Sẽ không có "lỗ" nào.
There are alternative objective functions. E.g., could have minimized sum of waste. This makes sense, especially if demand constraint is formulated as an inequality. Then minimizing sum of waste will spend more CPU cycles trying to find more efficient patterns that over-satisfy demand. This is especially good if demand widths recur regularly & storing cut rolls in inventory to satisfy future demand is possible. Note: running time will grow quickly with such an objective function.
– Có các hàm mục tiêu thay thế. Ví dụ, có thể giảm thiểu tổng lượng lãng phí. Điều này hợp lý, đặc biệt nếu ràng buộc nhu cầu được xây dựng dưới dạng bất đẳng thức. Khi đó, việc giảm thiểu tổng lượng lãng phí sẽ tốn nhiều chu kỳ CPU hơn để cố gắng tìm ra các mô hình hiệu quả hơn nhằm đáp ứng vượt mức nhu cầu. Điều này đặc biệt hữu ích nếu độ rộng nhu cầu lặp lại thường xuyên & việc lưu trữ các cuộn đã cắt trong kho để đáp ứng nhu cầu trong tương lai là khả thi. Lưu ý: thời gian chạy sẽ tăng nhanh với hàm mục tiêu như vậy.
Finally, rearrange solution to make sense for caller. Instead of our decision variables x, y, return a list of all rolls, with cutting patterns & waste of each.
– Cuối cùng, sắp xếp lại giải pháp để người gọi dễ hiểu hơn. Thay vì các biến quyết định x, y, hãy trả về danh sách tất cả các lần tung xúc xắc, kèm theo các mẫu cắt & lượng chất thải của mỗi lần tung.
Since need bounds on number of rolls & on maximum number of cuts satisfying a given order on a single roll, Listing 7.2: Cutting Stock Bounds Computation implements a simple heuristic. Minimum is clearly sum of all widths divided by width of a roll, 100, as assumed all widths to be entered as percentages. Upper bound is computed by a 1st fit heuristic: add each roll, in order, to a roll until no more fits. Then start a new roll. This is not brilliant, but it series its purpose.
– Vì giới hạn cần thiết là số lần cuộn & số lần cắt tối đa thỏa mãn 1 thứ tự nhất định trên 1 cuộn, Liệt kê 7.2: Tính toán Giới hạn Vật liệu Cắt triển khai 1 phương pháp tìm kiếm đơn giản. Giá trị nhỏ nhất rõ ràng là tổng của tất cả các chiều rộng chia cho chiều rộng của 1 cuộn, 100, với giả định tất cả các chiều rộng được nhập dưới dạng phần trăm. Giới hạn trên được tính bằng phương pháp tìm kiếm thứ nhất: cộng từng lần cuộn, theo thứ tự, vào 1 cuộn cho đến khi không còn lần nào khớp nữa. Sau đó, bắt đầu 1 lần cuộn mới. Phương pháp này không xuất sắc, nhưng nó thể hiện mục đích

của nó.

Listing 7.3: Cutting Stock Model Solution Post-Process reformats solution to make it more meaningful to caller. It returns an array containing for each roll used percentage of waste incurred on that particular roll as well as cut pattern used. Of course, indicated cut pattern could be permuted with no change to waste percentage. Output of our small example can be seen in Table 7.2: Optimal Solution to Cutting Stock.

– Liệt kê 7.3: Giải pháp Mô hình Cắt Nguyên liệu Hậu xử lý định dạng lại giải pháp để làm cho nó có ý nghĩa hơn đối với người gọi. Nó trả về 1 mảng chứa tỷ lệ phần trăm phế liệu phát sinh trên mỗi cuộn giấy cụ thể đó cũng như mẫu cắt được sử dụng. Tất nhiên, mẫu cắt được chỉ định có thể được hoán vị mà không thay đổi tỷ lệ phần trăm phế liệu. Kết quả của ví dụ nhỏ này có thể được xem trong Bảng 7.2: Giải pháp Tối ưu cho Cắt Nguyên liệu.

∗ 7.1.2. Pre-Allocate Cutting Patterns. Previous approach is optimal but will not scale very well, even with our symmetry-breaking constraints. I will describe here a generally suboptimal approach that can be used to solve much larger instances.

– Phân bổ Trước Các Mẫu Cắt. Cách tiếp cận trước đây là tối ưu nhưng sẽ không mở rộng tốt, ngay cả với các ràng buộc phá vỡ tính đối xứng của chúng ta. Tôi sẽ mô tả ở đây 1 cách tiếp cận nhìn chung không tối ưu nhưng có thể được sử dụng để giải quyết các trường hợp lớn hơn nhiều.

Basic idea: fix cutting patterns & only optimize number of rolls using those patterns while satisfying demand. Imagine, e.g., that we were given distinct patterns of last column of Table 7.2 in matrix A, as well as Table 7.1 in array D. Then we could have a decision variable y indexed by patterns of A, representing how many rolls to cut according to that pattern. Symbolically, we would have model

$$\min \sum_j y_j \ A_{j,i} y_j \geq D_i, \ \forall i, \ y_j \in \mathbb{N}.$$

It seems simple enough, until one carefully considers number of patterns. If do not know ahead of time which patterns to use, then simple option seems to be to list them all. How many are there? An astronomical number, even for small examples.

– Ý tưởng cơ bản: cố định các mẫu cắt & chỉ tối ưu hóa số lần cuộn bằng cách sử dụng các mẫu đó trong khi vẫn đáp ứng nhu cầu. Ví dụ, hãy tưởng tượng chúng ta được cung cấp các mẫu riêng biệt của cột cuối cùng trong Bảng 7.2 trong ma trận A, cũng như Bảng 7.1 trong mảng D. Khi đó, chúng ta có thể có 1 biến quyết định y được lập chỉ mục theo các mẫu của A, biểu thị số lần cuộn cần cắt theo mẫu đó. Về mặt biểu tượng, chúng ta sẽ có mô hình

$$\min \sum_j y_j \ A_{j,i} y_j \geq D_i, \ \forall i, \ y_j \in \mathbb{N}.$$

Nghe có vẻ đơn giản, cho đến khi chúng ta cân nhắc kỹ lưỡng số lượng mẫu. Nếu không biết trước nên sử dụng mẫu nào, thì lựa chọn đơn giản dường như là liệt kê tất cả. Có bao nhiêu mẫu? 1 con số khổng lồ, ngay cả đối với những ví dụ nhỏ. So, here is brilliant idea: start with a certain set of patterns, optimize. Then, in a manner yet to be determined, find some "better" patterns to add. In all its generality, this is known to optimizers [Expression column generation stemmed, in minds of optimizers, as literal-minded as ever, from looking at set of constraints as a matrix.] as *column generation*. Repeat optimization until we can no longer find "better" patterns or until we run out of time or are satisfied with solution. Listing 7.4: Cutting Stock Model Using Given Patterns implements high-level approach.

– Vậy, đây là 1 ý tưởng tuyệt vời: bắt đầu với 1 tập hợp các mẫu nhất định, tối ưu hóa. Sau đó, theo 1 cách chưa được xác định, tìm 1 số mẫu "tốt hơn" để thêm vào. Về tổng quát, điều này được các nhà tối ưu hóa biết đến [Việc tạo cột biểu thức bắt nguồn, trong tâm trí của các nhà tối ưu hóa, vẫn theo nghĩa đen như mọi khi, từ việc xem xét tập hợp các ràng buộc như 1 ma trận.] như là *tạo cột*. Lặp lại quá trình tối ưu hóa cho đến khi chúng ta không thể tìm thấy các mẫu "tốt hơn" nữa hoặc cho đến khi hết thời gian hoặc hài lòng với giải pháp. Liệt kê 7.4: Mô hình Cắt Cổ phiếu Sử dụng Các Mẫu Cho sẵn triển khai phương pháp tiếp cận cấp cao.

At line 4, loop on optimization a certain number of times. This is an easy termination criteria, which we can tune according to size of problem we are solving & how long we are willing to wait. There are better criteria, including looping until we have a true optimal solution, but they would lead us deep into theory of optimization. [Interested reader should look up "reduced cost" & "column generation of cutting stock" to pursue these matters.]

– Ở dòng 4, lặp lại tối ưu hóa 1 số lần nhất định. Đây là 1 tiêu chí kết thúc dễ dàng, chúng ta có thể điều chỉnh tùy theo quy mô của vấn đề đang giải quyết & thời gian chúng ta sẵn sàng chờ đợi. Có những tiêu chí tốt hơn, bao gồm cả việc lặp lại cho đến khi chúng ta có 1 giải pháp tối ưu thực sự, nhưng chúng sẽ dẫn chúng ta đi sâu vào lý thuyết tối ưu. [Bạn đọc quan tâm nên tra cứu "giảm chi phí" & "tạo cột vật liệu cắt" để tìm hiểu thêm về những vấn đề này.]

Master problem is essentially the one described above: given set of allowable patterns, do your best to minimize number of rolls. There are 2 subtleties. 1st one: we solve optimization problem as a linear program, not an integer program, even though we really want an integer solution, number of rolls. Do this to gain speed. At end, simply round up number of rolls, since obviously if 4.6 rolls will satisfy demand, then surely 5 rolls will also satisfy demand.

– Bài toán chính về cơ bản là bài toán được mô tả ở trên: cho trước 1 tập hợp các mẫu cho phép, hãy cố gắng hết sức để giảm thiểu số lần tung. Có 2 điểm tinh tế. Thứ nhất: chúng ta giải bài toán tối ưu hóa bằng 1 chương trình tuyến tính, không phải chương trình số nguyên, mặc dù chúng ta thực sự muốn 1 nghiệm số nguyên, tức là số lần tung. Hãy làm như vậy để tăng tốc độ. Cuối cùng, chỉ cần làm tròn số lần tung, vì rõ ràng nếu 4,6 lần tung sẽ đáp ứng nhu cầu, thì chắc chắn 5 lần tung cũng sẽ đáp ứng nhu cầu.

2nd subtlety involves information we need to find better patterns. Consider 1 imaginary example of a constraint of form of line 21: for, say product roll 5, imagine we need 28 consumer rolls to satisfy demand. Given patterns we have already,

we might have that pattern 1 has this roll 3 times, pattern 3 has it 5 times, & pattern 10 has it once (& no other pattern has roll 5). So constraint reads $3y_1 + 5y_3 + 1y_{10} \geq 28$, where solution is y. What is effect of changing 28 by exactly 1 unit, keeping everything else constant? It will change optimal solution by a small value, marginal value of roll 5. We can do this, conceptually, for every roll & get each of their marginal values. By design, all solvers already have these marginal values computed; they are a side-effect of solution techniques. So simply request them at line 25 by call for `DualValues`.

– Sự tinh tế thứ hai liên quan đến thông tin chúng ta cần để tìm ra các mẫu tốt hơn. Hãy xem xét 1 ví dụ tưởng tượng về ràng buộc dạng dòng 21: giả sử với lần tung sản phẩm thứ 5, hãy tưởng tượng chúng ta cần 28 lần tung tiêu dùng để đáp ứng nhu cầu. Với các mẫu đã có, chúng ta có thể có mẫu 1 tung sản phẩm này 3 lần, mẫu 3 tung sản phẩm này 5 lần, & mẫu 10 tung sản phẩm này 1 lần (& không có mẫu nào khác tung sản phẩm này lần thứ 5). Vì vậy, ràng buộc là $3y_1 + 5y_3 + 1y_{10} \geq 28$, với nghiệm là y. Việc thay đổi 28 đúng 1 đơn vị, giữ nguyên mọi thứ khác, sẽ ảnh hưởng như thế nào đến nghiệm tối ưu? Nó sẽ thay đổi nghiệm tối ưu 1 giá trị nhỏ, giá trị biên của lần tung thứ 5. Về mặt khái niệm, chúng ta có thể làm điều này với mỗi lần tung sản phẩm & lấy từng giá trị biên của chúng. Theo thiết kế, tất cả các trình giải đều đã tính toán các giá trị biên này; chúng là 1 tác dụng phụ của các kỹ thuật giải. Vì vậy, chỉ cần yêu cầu chúng ở dòng 25 bằng cách gọi `DualValues`.

At end, reformat solution to make it meaningful to caller. Return an array containing for each roll used, waste incurred on that particular roll, & cut pattern used.

– Cuối cùng, hãy định dạng lại giải pháp để người gọi dễ hiểu hơn. Trả về 1 mảng chứa lượng chất thải phát sinh trên mỗi lần tung xúc xắc, & mẫu cắt đã sử dụng.

Model to find a new pattern for master model to optimize over (Listing 7.5: Cutting Stock Model (Getting a New Pattern)) uses marginal value of each roll, provided by solution to master model above & maximizes sum of values times number of occurrence of that roll in a pattern, while ensuring that pattern stays within total width of large roll at line 7. This is a knapsack problem & will be solved very fast.

– Mô hình để tìm 1 mẫu mới cho mô hình chính nhằm tối ưu hóa (Liệt kê 7.5: Mô hình cắt phôi (Lấy mẫu mới)) sử dụng giá trị cận biên của mỗi lần cuộn, được cung cấp bởi giải pháp cho mô hình chính ở trên & tối đa hóa tổng giá trị nhân với số lần xuất hiện của lần cuộn đó trong 1 mẫu, đồng thời đảm bảo rằng mẫu đó nằm trong tổng chiều rộng của cuộn lớn tại dòng 7. Đây là bài toán về ba lô & sẽ được giải quyết rất nhanh.

We have left 2 elements undescribed: initial patterns, & reshaping of solution to make it meaningful. From Listing 7.6: Cutting Stock Model (Getting a New Pattern), initial patterns must be s.t. they will allow a feasible solution, one that satisfies all demands. Could be very creative here, or not. Considering already complex model, go latter route. Our initial patterns have exactly 1 roll per pattern, as obviously feasible as inefficient.

– Chúng ta đã bỏ sót 2 yếu tố chưa được mô tả: các mẫu ban đầu, & việc định hình lại giải pháp để làm cho nó có ý nghĩa. Từ Liệt kê 7.6: Mô hình Cắt Cổ phiếu (Lấy Mẫu Mới), các mẫu ban đầu phải được s.t. chúng sẽ cho phép 1 giải pháp khả thi, 1 giải pháp đáp ứng mọi yêu cầu. Có thể rất sáng tạo ở đây, hoặc không. Xem xét mô hình đã phức tạp, hãy đi theo hướng sau. Các mẫu ban đầu của chúng ta có đúng 1 lần lăn cho mỗi mẫu, vừa khả thi vừa không hiệu quả.

A solution to our small example with this column generation approach is shown in Table 7.3: Possibly Sub-Optimal Solution to Cutting Stock Using Column Generation. Note: it may use more rolls, but it actually cuts each roll very efficiently. It simply over-satisfies demands, unsurprising since we rounded up.

– Giải pháp cho ví dụ nhỏ của chúng ta với phương pháp tạo cột này được thể hiện trong Bảng 7.3: Giải pháp có thể chưa tối ưu để cắt vật liệu bằng phương pháp tạo cột. Lưu ý: phương pháp này có thể sử dụng nhiều cuộn hơn, nhưng thực tế nó cắt mỗi cuộn rất hiệu quả. Nó chỉ đơn giản là đáp ứng quá mức nhu cầu, điều này không có gì đáng ngạc nhiên vì chúng ta đã làm tròn số.

3.3 [Pin22]. Michael L. Pinedo. Scheduling: Theory, Algorithms, & Systems. 6e

- Preface to 6e. Since release of 1e in 1994, scheduling field has seen many new developments that are of interest to theoreticians & practitioners alike. Clearly, new editions of book, with extensions in different directions, were to be expected. However, basic general setup & structure of book has not changed over years. It still consists of 3 main parts: Deterministic Models, Stochastic Models, & Scheduling in Practice. There are also 4 Appendices that present basics of mathematical programming, dynamic programming, constraint programming, & complexity theory, as well as 3 Appendixes that provide overviews of complexity statuses of several classes of deterministic scheduling problems, of tractability of a variety of stochastic scheduling problems, & of latest developments in scheduling system designs & implementations.

  Since its introduction in 1994 this book has undergone a number of expansions, with extensions in areas that have aroused interest in academia as well as industry. These extensions have included over years multi-objective scheduling, batch scheduling, & proportionate flow shop scheduling. Additions in this 6e include deterministic flow shop models with reentry (i.e., scheduling models that are of interest to semiconductor manufacturing industry), stochastic models with due date related objective functions, Fixed Parameter Tractability (FPT) of deterministic scheduling problems, as well as discussions regarding recent scheduling system implementations. Latest updates of various tables in Appendixes E, F, G have been extensive.

  Part I of book (Deterministic Models) can still be used as basis for a course in deterministic scheduling at a Senior or Masters level in an Engineering school or in an Applied Mathematics department. Parts I & II together (Deterministic & Stochastic Models) can be used as a basis for a Masters or PhD level course in an engineering or a business school.

  A solution manual is still available. However, because of specific requests made by several of faculty who have contributed to contents of this manual, it can only be sent out to instructors who actually teach a course at an established university. There is also a fair amount of supplementary material available, closely related to content of book (e.g., PowerPoint presentations, scheduling cases, etc.), that can be downloaded from author's homepage at wp.nyu.edu/michaelpinedo/books/.

- Gantt Chart: Its Originators. Every practitioner & researcher in scheduling field has used Gantt charts as a scheduling tool. Origins of these famous charts are interesting; they were developed independently in Poland & in US in late 19th & early 20th century.

  In 1896 Polish engineer & economist Karol Adamiecki developed what he called *harmonogram*, which he used in management of a steel rolling mill in southern Poland. Adamiecki published his techniques in Polish magazine *Przeglad Techniczny* (Technical Review) in 1909.

  Shortly thereafter, Henry Laurence Gantt, an industrial engineer working in steel industry in US, developed his charts for evaluating production schedules. Gantt discussed principles underlying his charts in his book "Organizing for Work", which was published just before his death in 1919. Since Gantt communicated in English & Adamiecki in Polish, these types of charts are in English literature usually referred to as Gantt charts.

  Charts currently being used in real world decision support systems are at times somewhat different from originals developed by Adamiecki & Gantt, in their design as well as in their purpose.

- Supplementary Electronic Material. Supplementary electronic material listed below is available for download from author's homepage at wp.nyu.edu/michaelpinedo/books/: slides from Academia, scheduling systems: LEKIN, LiSA, TORSCHE, scheduling case: scheduling in time-shared jet business, mini-cases, handouts, movies: SAIGA – scheduling at Paris Airports (ILOG), scheduling at United Airlines, preactor international.

- 1. Introduction.

  - 1.1. Role of Scheduling. Scheduling is a decision-making process that is used on a regular basis in many manufacturing & services industries. It deals with allocation of resources to tasks over given time periods & its goal is to optimize 1 or more objectives.

    – *Vai trò của Lập lịch.* Lập lịch là 1 quá trình ra quyết định được sử dụng thường xuyên trong nhiều ngành sản xuất & dịch vụ. Nó liên quan đến việc phân bổ nguồn lực cho các nhiệm vụ trong khoảng thời gian nhất định & mục tiêu của nó là tối ưu hóa 1 hoặc nhiều mục tiêu.

    Resources & tasks in an organization can take many different forms. Resources may be machines in a workshop, runways at an airport, crews at a construction site, processing units in a computing environment, etc. Tasks may be operations in a production process, take-offs & landings at an airport, stages in a construction project, executions of computer programs, etc. Each task may have a certain priority level, an earliest possible starting time, & a due date. Objectives can also take many different forms. 1 objective may be minimization of completion time of last tasks & another may be minimization of number of tasks completed after their respective due dates.

    – Tài nguyên & nhiệm vụ trong 1 tổ chức có thể có nhiều hình thức khác nhau. Tài nguyên có thể là máy móc trong xưởng, đường băng tại sân bay, phi hành đoàn tại công trường xây dựng, đơn vị xử lý trong môi trường máy tính, v.v. Nhiệm vụ có thể là các hoạt động trong quy trình sản xuất, cất cánh & hạ cánh tại sân bay, các giai đoạn trong dự án xây dựng, thực hiện các chương trình máy tính, v.v. Mỗi nhiệm vụ có thể có 1 mức độ ưu tiên nhất định, thời gian bắt đầu sớm nhất có thể, & ngày đến hạn. Mục tiêu cũng có thể có nhiều hình thức khác nhau. 1 mục tiêu có thể là giảm thiểu thời gian hoàn thành của các nhiệm vụ cuối cùng & mục tiêu khác có thể là giảm thiểu số lượng nhiệm vụ hoàn thành sau ngày đến hạn tương ứng của chúng.

Scheduling, as a decision-making process, plays an important role in most manufacturing & production systems as well as in most information processing environments. Also important in transportation & distribution settings & in other types of service industries. Following examples illustrate role of scheduling in a number of real-world environments.

– Lên lịch, như 1 quá trình ra quyết định, đóng vai trò quan trọng trong hầu hết các hệ thống sản xuất & cũng như trong hầu hết các môi trường xử lý thông tin. Cũng quan trọng trong các thiết lập vận chuyển & phân phối & trong các loại ngành dịch vụ khác. Các ví dụ sau minh họa vai trò của việc lên lịch trong 1 số môi trường thực tế.

Example 1 (A paper bag factory)**.** *Consider a factory that produces paper bags for cement, charcoal, dog food, etc. Basic raw material for such an operation are rolls of paper. Production process consists of 3 stages: printing of logo, gluing of side of bag, & sewing of 1 end or both ends of bag. Each stage consists of a number of machines which are not necessarily identical. Machines at a stage may differ slightly in speed at which they operate, number of colors they can print, or size of bag they can produce. Each production order indicates a given quantity of a specific bag that has to be produced & shipped by a committed shipping date or due date. Processing time for different operations are proportional to size of order, i.e., number of bags ordered.*

– Hãy xem xét 1 nhà máy sản xuất túi giấy đựng xi măng, than củi, thức ăn cho chó, v.v. Nguyên liệu thô cơ bản cho hoạt động như vậy là cuộn giấy. Quy trình sản xuất bao gồm 3 công đoạn: in logo, dán mép túi, & may 1 đầu hoặc cả hai đầu túi. Mỗi công đoạn bao gồm 1 số máy không nhất thiết phải giống hệt nhau. Các máy ở 1 công đoạn có thể hơi khác nhau về tốc độ hoạt động, số lượng màu có thể in hoặc kích thước túi có thể sản xuất. Mỗi lệnh sản xuất chỉ ra số lượng nhất định của 1 loại túi cụ thể phải được sản xuất & vận chuyển theo ngày giao hàng hoặc ngày đến hạn đã cam kết. Thời gian xử lý cho các hoạt động khác nhau tỷ lệ thuận với quy mô đơn hàng, tức là số lượng túi đã đặt hàng.

A late delivery implies a penalty in form of loss of goodwill & magnitude of penalty depends on importance of order or client & tardiness of delivery. 1 of objectives of scheduling system is to minimize sum of these penalties.

– Giao hàng trễ sẽ phải chịu hình phạt dưới hình thức mất uy tín & mức độ phạt phụ thuộc vào tầm quan trọng của đơn hàng hoặc khách hàng & thời gian giao hàng chậm. 1 trong những mục tiêu của hệ thống lập lịch là giảm thiểu tổng số tiền phạt này.

When a machine is switched over from 1 type of bag to another, a setup is required. Length of setup time on machine depends on similarities between 2 consecutive orders (number of colors in common, differences in bag size, etc.). An important objective of scheduling system is minimization of total time spent on setups.

– Khi máy được chuyển từ loại túi này sang loại túi khác, cần phải thiết lập. Thời gian thiết lập trên máy phụ thuộc vào điểm tương đồng giữa 2 đơn hàng liên tiếp (số lượng màu chung, sự khác biệt về kích thước túi, v.v.). 1 mục tiêu quan trọng của hệ thống lập lịch là giảm thiểu tổng thời gian dành cho việc thiết lập.

Example 2 (A semiconductor manufacturing facility)**.** *Semiconductors are manufactured in highly specialized facilities. This is case with memory chips as well as with microprocessors. Production process in these facilities usually consists of 4 phases: wafer fabrication, wafer probe, assembly or packaging, & final testing.*

– Chất bán dẫn được sản xuất tại các cơ sở chuyên dụng cao. Trường hợp này cũng xảy ra với chip nhớ cũng như với bộ vi xử lý. Quy trình sản xuất tại các cơ sở này thường bao gồm 4 giai đoạn: chế tạo wafer, đầu dò wafer, lắp ráp hoặc đóng gói, & thử nghiệm cuối cùng.

Wafer fabrication is technologically the most complex phase. Layers of metal & wafer material are built up in patterns on wafers of silicon or gallium arsenide to produce circuitry. Each layer requires a number of operations, which typically include (i) cleaning, (ii) oxidation, deposition, & metallization, (iii) lithography, (iv) etching, (v) ion implantation, (vi) photoresist stripping, (vii) inspection & measurement. Because it consists of various layers, each wafer has to undergo these operations several times. Thus, there is a significant amount of recirculation in process. Wafers move through facility in lots of 24*. Some machines may require setups to prepare them for incoming jobs; setup time often depends on configurations of lot just completed & lot about to start.*

– Chế tạo wafer là giai đoạn phức tạp nhất về mặt công nghệ. Các lớp kim loại & vật liệu wafer được tạo thành các mẫu trên wafer silicon hoặc gali arsenide để tạo ra mạch điện. Mỗi lớp yêu cầu 1 số thao tác, thường bao gồm (i) làm sạch, (ii) oxy hóa, lắng đọng, & kim loại hóa, (iii) quang khắc, (iv) khắc, (v) cấy ion, (vi) loại bỏ chất cản quang, (vii) kiểm tra & đo lường. Vì bao gồm nhiều lớp khác nhau, mỗi wafer phải trải qua các thao tác này nhiều lần. Do đó, có 1 lượng tuần hoàn đáng kể trong quá trình này. Các wafer di chuyển qua cơ sở theo từng lô 24*. 1 số máy có thể yêu cầu thiết lập để chuẩn bị cho các công việc sắp tới; thời gian thiết lập thường phụ thuộc vào cấu hình của lô vừa hoàn thành & lô sắp bắt đầu.*

Number of orders in production process is often in hundreds & each has its own release date & a committed shipping or due date. Scheduler's objective: meet as many of committed shipping dates as possible, which maximizing throughput. Latter goal is achieved by maximizing equipment utilization, especially of bottleneck machines, requiring thus a minimization of idle times & setup times.

– Số lượng đơn hàng trong quá trình sản xuất thường lên tới hàng trăm & mỗi đơn hàng có ngày phát hành riêng & ngày giao hàng hoặc ngày đến hạn đã cam kết. Mục tiêu của người lập lịch: đáp ứng càng nhiều ngày giao hàng đã cam kết càng tốt, qua đó tối đa hóa thông lượng. Mục tiêu sau đạt được bằng cách tối đa hóa việc sử dụng thiết bị, đặc biệt là các máy móc bị tắc nghẽn, do đó đòi hỏi phải giảm thiểu thời gian nhàn rỗi & thời gian thiết lập.

Example 3 (Gate assignments at an airport)**.** *Consider an airline terminal at a major airport. There are dozens of gates & hundreds of planes arriving & departing each day. Gates are not all identical & neither are planes. Some of gates are in locations with a lot of space where large planes (widebodies) can be accommodated easily. Other gates are in locations where it is difficult to bring in the planes; certain planes may actually have to be towed to their gates.*

– Hãy xem xét 1 nhà ga hàng không tại 1 sân bay lớn. Có hàng chục cổng & hàng trăm máy bay đến & khởi hành mỗi ngày. Không phải tất cả các cổng đều giống nhau & máy bay cũng vậy. 1 số cổng nằm ở những vị trí có nhiều không gian, nơi có thể dễ dàng chứa được những chiếc máy bay lớn (máy bay thân rộng). Những cổng khác nằm ở những vị trí khó đưa máy bay vào; 1 số máy bay thực sự có thể phải được kéo đến cổng của chúng.

Planes arrive & depart according to a certain schedule. However, schedule is subject to a certain amount of randomness, which may be weather related or caused by unforeseen events at other airports. During time that a plane occupies a gate the arriving passengers have to be deplaned, plane has to be serviced, & departing passengers have to be boarded. Scheduled departure time can be viewed as a due date & airline's performance is measured accordingly. However, if it is known in advance: plane cannot land at next airport because of anticipated congestion at its scheduled arrival time, then plane does not take off (such a policy is followed to conserve fuel). If a plane is not allowed to take off, operating policies usually prescribe that passengers remain in terminal rather than on plane. If boarding is postponed, a plane may remain at a gate for an extended period of time, thus preventing other planes from using that gate.

– Máy bay đến & khởi hành theo 1 lịch trình nhất định. Tuy nhiên, lịch trình phụ thuộc vào 1 lượng ngẫu nhiên nhất định, có thể liên quan đến thời tiết hoặc do các sự kiện không lường trước được tại các sân bay khác. Trong thời gian máy bay chiếm 1 cổng, hành khách đến phải xuống máy bay, máy bay phải được phục vụ, & hành khách khởi hành phải lên máy bay. Thời gian khởi hành theo lịch trình có thể được xem là ngày đến hạn & hiệu suất của hãng hàng không được đo lường theo đó. Tuy nhiên, nếu biết trước: máy bay không thể hạ cánh tại sân bay tiếp theo do dự đoán có tình trạng tắc nghẽn tại thời gian đến theo lịch trình, thì máy bay sẽ không cất cánh (chính sách như vậy được áp dụng để tiết kiệm nhiên liệu). Nếu máy bay không được phép cất cánh, các chính sách khai thác thường quy định hành khách phải ở lại nhà ga thay vì trên máy bay. Nếu việc lên máy bay bị hoãn, máy bay có thể ở lại 1 cổng trong 1 thời gian dài, do đó ngăn không cho các máy bay khác sử dụng cổng đó.

Scheduler has to assign planes to gates in such a way that assignment is physically feasible while optimizing a number of objectives. This implies: scheduler has to assign planes to suitable gates available at respective arrival times. Objective include minimization of work for airline personnel & minimization of airplane delays.

– Người lập lịch phải chỉ định máy bay đến các cổng theo cách mà việc chỉ định khả thi về mặt vật lý trong khi tối ưu hóa 1 số mục tiêu. Điều này ngụ ý: người lập lịch phải chỉ định máy bay đến các cổng phù hợp có sẵn tại thời điểm đến tương ứng. Mục tiêu bao gồm giảm thiểu công việc cho nhân viên hãng hàng không & giảm thiểu sự chậm trễ của máy bay.

In this scenario gates are resources & handling & servicing of planes are tasks. Arrival of a plane at a gate represents starting time of a tasks & departure represents its completion time.

– Trong kịch bản này, cổng là tài nguyên & xử lý & bảo dưỡng máy bay là nhiệm vụ. Máy bay đến cổng biểu thị thời gian bắt đầu của nhiệm vụ & khởi hành biểu thị thời gian hoàn thành.

Example 4 (Scheduling tasks in a central processing unit (CPU))**.** *1 of functions of a multi-tasking computer OS: schedule time CPU devotes to different programs that have to be executed. Exact processing times are usually not known in advance. However, distribution of these random processing times may be known in advance, including their means & their variances. In addition, each task usually has a certain priority level (OS typically allows operators & users to specify priority level or weight of each task). In such a case, objective: minimize expected sum of weighted completion times of all tasks.*

– 1 trong những chức năng của máy tính đa nhiệm HDH: thời gian lập lịch CPU dành cho các chương trình khác nhau phải được thực thi. Thời gian xử lý chính xác thường không được biết trước. Tuy nhiên, sự phân bố của các thời gian xử lý ngẫu nhiên này có thể được biết trước, bao gồm cả phương tiện & phương sai của chúng. Ngoài ra, mỗi tác vụ thường có 1 mức độ ưu tiên nhất định (HDH thường cho phép người vận hành & người dùng chỉ định mức độ ưu tiên hoặc trọng số của từng tác vụ). Trong trường hợp như vậy, mục tiêu: giảm thiểu tổng thời gian hoàn thành có trọng số dự kiến của tất cả các tác vụ.

To avoid situation where relatively short tasks remain in system for a long time waiting for much longer tasks that have a higher priority, OS "slides" each task into little pieces. OS then rotates these slices on CPU so that in any given time interval, CPU spends some amount of time on each task. This way, if by chance processing time of 1 of tasks is very short, task will be able to leave system relatively quickly.

– Để tránh tình huống các tác vụ tương đối ngắn vẫn nằm trong hệ thống trong thời gian dài chờ đợi các tác vụ dài hơn nhiều có mức độ ưu tiên cao hơn, HDH "trượt" từng tác vụ thành các phần nhỏ. Sau đó, HDH xoay các phần này trên CPU để trong bất kỳ khoảng thời gian nào, CPU dành 1 khoảng thời gian nhất định cho từng tác vụ. Theo cách này, nếu thời gian xử lý của 1 trong các tác vụ rất ngắn, tác vụ sẽ có thể rời khỏi hệ thống tương đối nhanh.

An interruption of processing of a tasks is often referred to as a preemption. Clear: optimal policy in such an environment makes heavy use of preemptions.

– Việc gián đoạn xử lý tác vụ thường được gọi là quyền ưu tiên. Rõ ràng: chính sách tối ưu trong môi trường như vậy sử dụng nhiều quyền ưu tiên.

It may not be immediately clear what impact schedules may have on objectives of interest. Does it make sense to invest time & effort searching for a good schedule rather than just choosing a schedule at random? In practice, it often turns out: choice of schedule *does* have a significant impact on system's performance & it *does* make sense to spend some time & effort searching for a suitable schedule.

– Có thể không rõ ràng ngay lập tức về tác động của lịch trình lên các mục tiêu quan tâm. Có hợp lý không khi đầu tư thời gian & công sức tìm kiếm 1 lịch trình tốt thay vì chỉ chọn 1 lịch trình ngẫu nhiên? Trong thực tế, thường thì: lựa chọn lịch trình có tác động đáng kể đến hiệu suất của hệ thống & việc dành thời gian & công sức tìm kiếm 1 lịch trình phù hợp là hợp lý.

Scheduling can be difficult from a technical as well as an implementation point of view. Type of difficulties encountered on technical side are similar to difficulties encountered in other forms of combinatorial optimization & stochastic modeling. Difficulties on implementation side are of a completely different kind. They may depend on accuracy of model used for analysis of actual scheduling problem & on reliability of input data needed.

– Lên lịch có thể khó khăn từ góc độ kỹ thuật cũng như góc độ triển khai. Loại khó khăn gặp phải về mặt kỹ thuật tương tự như khó khăn gặp phải trong các hình thức tối ưu hóa tổ hợp khác & mô hình ngẫu nhiên. Khó khăn về mặt triển khai có loại hoàn toàn khác. Chúng có thể phụ thuộc vào độ chính xác của mô hình được sử dụng để phân tích vấn đề lập lịch thực tế & độ tin cậy của dữ liệu đầu vào cần thiết.

- 1.2. Scheduling Function in an Enterprise. Scheduling function in a production system or service organization must interact with many other functions. These interactions are system-dependent & may differ substantially from 1 situation to another. They often take place within an enterprise-wide information system.

  – Chức năng lập lịch trong 1 hệ thống sản xuất hoặc tổ chức dịch vụ phải tương tác với nhiều chức năng khác. Những tương tác này phụ thuộc vào hệ thống & có thể khác nhau đáng kể tùy từng tình huống. Chúng thường diễn ra trong 1 hệ thống thông tin toàn doanh nghiệp.

  A modern factory or service organization often has an elaborate information system in place that includes a central computer & database. Local area networks of personal computers, workstations, & data entry terminals, which are connected to this central computer, may be used either to retrieve data from database or to enter new data. Software controlling e.g. elaborate information system is typically referred to as an Enterprise Resource Planning (ERP) system. A number of software companies specialize in development of such systems, including SAP, J.D. Edwards, & PeopleSoft. Such an ERP system plays role of an information highway that traverses enterprise with, at al organizational levels, links to decision support systems.

  – 1 nhà máy hiện đại hoặc tổ chức dịch vụ thường có 1 hệ thống thông tin phức tạp bao gồm 1 máy tính trung tâm & cơ sở dữ liệu. Mạng cục bộ của máy tính cá nhân, máy trạm, & thiết bị đầu cuối nhập dữ liệu, được kết nối với máy tính trung tâm này, có thể được sử dụng để truy xuất dữ liệu từ cơ sở dữ liệu hoặc để nhập dữ liệu mới. Phần mềm kiểm soát ví dụ như hệ thống thông tin phức tạp thường được gọi là hệ thống Lập kế hoạch nguồn lực doanh nghiệp (ERP). 1 số công ty phần mềm chuyên phát triển các hệ thống như vậy, bao gồm SAP, J.D. Edwards, & PeopleSoft. 1 hệ thống ERP như vậy đóng vai trò là 1 xa lộ thông tin đi qua doanh nghiệp với, ở mọi cấp độ tổ chức, liên kết đến các hệ thống hỗ trợ quyết định.

  Scheduling is often done interactively via a decision support system installed on a personal computer or workstation linked to ERP system. Terrminals at key locations connected to ERP system can give departments throughout enterprise access to all current scheduling information. These departments, in turn, can provide scheduling system with up-to-date information concerning statuses of jobs & machines.

  – Việc lập lịch thường được thực hiện tương tác thông qua hệ thống hỗ trợ quyết định được cài đặt trên máy tính cá nhân hoặc máy trạm được liên kết với hệ thống ERP. Các thiết bị đầu cuối tại các vị trí quan trọng được kết nối với hệ thống ERP có thể cung cấp cho các phòng ban trong toàn doanh nghiệp quyền truy cập vào tất cả thông tin lập lịch hiện tại. Đổi lại, các phòng ban này có thể cung cấp cho hệ thống lập lịch thông tin cập nhật liên quan đến trạng thái của công việc & máy móc.

  There are, of course, still environments where communication between scheduling function & other decision-making entities occurs in meetings or through memos.

  – Tất nhiên, vẫn có những môi trường mà việc giao tiếp giữa chức năng lập lịch trình & các thực thể ra quyết định khác diễn ra trong các cuộc họp hoặc thông qua bản ghi nhớ.

  * Scheduling in Manufacturing. Consider following generic manufacturing environment & role of its scheduling. Orders that are released in a manufacturing setting have to be translated into jobs with associated due dates. These jobs often have to be processed on machines in a workcenter in a given order or sequence. Processing of jobs may sometimes be delayed if certain machines are busy & preemptions may occur when high-priority jobs arrive at machines that are busy. Unforeseen events on shop floor, e.g. machine breakdowns or longer-than-expected processing times, also have to be taken into account, since they may have a major impact on schedules. In such an environment, development of a detailed task schedule helps maintain efficiency & control of operations.

    – *Lên lịch trong sản xuất.* Hãy cân nhắc đến môi trường sản xuất chung sau & vai trò của lịch trình. Các đơn hàng được phát hành trong môi trường sản xuất phải được chuyển thành các công việc có ngày đến hạn liên quan. Những công việc này thường phải được xử lý trên các máy trong 1 trung tâm làm việc theo thứ tự hoặc trình tự nhất định. Việc xử lý các công việc đôi khi có thể bị chậm trễ nếu 1 số máy nhất định đang bận & có thể xảy ra tình trạng chiếm dụng trước khi các công việc có mức độ ưu tiên cao đến với các máy đang bận. Các sự kiện không lường trước được trên sàn nhà máy, ví dụ như máy hỏng hoặc thời gian xử lý dài hơn dự kiến, cũng phải được tính đến, vì chúng có thể ảnh hưởng lớn đến lịch trình. Trong môi trường như vậy, việc phát triển 1 lịch trình công việc chi tiết giúp duy trì hiệu quả & kiểm soát hoạt động.

    Shop floor is not only part of organization that impacts scheduling process. It is also affected by production planning process that handles medium- to long-term planning for entire organization. This process attempts to optimize firm's overall product mix & long-term resource allocation based on its inventory levels, demand forecasts, & resource requirements. Decisions made at this higher planning level may impact scheduling process directly. Fig. 1.1: Information flow diagram in a manufacturing system depicts a diagram of information flow in a manufacturing system.

    – Xưởng sản xuất không chỉ là 1 phần của tổ chức tác động đến quy trình lập lịch trình. Nó cũng bị ảnh hưởng bởi quy trình lập kế hoạch sản xuất xử lý kế hoạch trung hạn đến dài hạn cho toàn bộ tổ chức. Quy trình này cố gắng tối ưu hóa

hỗn hợp sản phẩm tổng thể của công ty & phân bổ nguồn lực dài hạn dựa trên mức tồn kho, dự báo nhu cầu, & yêu cầu về nguồn lực. Các quyết định được đưa ra ở cấp độ lập kế hoạch cao hơn này có thể tác động trực tiếp đến quy trình lập lịch trình. Hình 1.1: Sơ đồ luồng thông tin trong hệ thống sản xuất mô tả sơ đồ luồng thông tin trong hệ thống sản xuất.

In a manufacturing environment, scheduling function has to interact with other decision-making functions. 1 popular system that is widely used is Material Requirements Planning (MRP) system. After a schedule has been generated, necessary: all raw materials & resources are available at specified times. Ready dates of all jobs have to be determined jointly by production planning/scheduling system & MRP system.

– Trong môi trường sản xuất, chức năng lập lịch phải tương tác với các chức năng ra quyết định khác. 1 hệ thống phổ biến được sử dụng rộng rãi là hệ thống Lập kế hoạch yêu cầu vật liệu (MRP). Sau khi lập lịch, cần: tất cả nguyên vật liệu & tài nguyên đều có sẵn tại thời điểm cụ thể. Ngày sẵn sàng của tất cả các công việc phải được xác định chung bởi hệ thống lập kế hoạch sản xuất/lập lịch & hệ thống MRP.

MRP systems are normally fairly elaborate. Each job has a Bill of Materials (BOM) itemizing parts required for production. MRP system keeps track of inventory of each part. Furthermore, it determines timing of purchases of each 1 of materials. In doing so, it uses techniques e.g. lot sizing & lot scheduling that are similar to those used in scheduling systems. There are many commercial MRP software packages available &, as a result, there are many manufacturing facilities with MRP systems. In cases where facility does not have a scheduling system, MRP system may be used for production planning purposes. However, in complex settings, not easy for an MRP system to do detailed scheduling satisfactorily.

– Hệ thống MRP thường khá phức tạp. Mỗi công việc đều có 1 Danh mục vật liệu (BOM) liệt kê các bộ phận cần thiết cho sản xuất. Hệ thống MRP theo dõi hàng tồn kho của từng bộ phận. Hơn nữa, nó xác định thời điểm mua từng loại vật liệu. Khi làm như vậy, nó sử dụng các kỹ thuật ví dụ như định cỡ lô & lập lịch lô tương tự như các kỹ thuật được sử dụng trong hệ thống lập lịch. Có nhiều gói phần mềm MRP thương mại có sẵn & do đó, có nhiều cơ sở sản xuất có hệ thống MRP. Trong trường hợp cơ sở không có hệ thống lập lịch, hệ thống MRP có thể được sử dụng cho mục đích lập kế hoạch sản xuất. Tuy nhiên, trong các bối cảnh phức tạp, không dễ để hệ thống MRP lập lịch chi tiết 1 cách thỏa đáng.

∗ Scheduling in Services. Describing a generic service organization & a typical scheduling system is not as easy as describing a generic manufacturing organization. Scheduling function in a service organization may face a variety of problems. It may have to deal with reservation of resources, e.g., assignment of planes to gates, or reservation of meeting rooms or other facilities. Models used are at times somewhat different from those used in manufacturing settings. Scheduling in a service environment must be coordinated with other decision-making functions, usually within elaborate information systems, much in same way as scheduling function in a manufacturing setting. These information systems usually rely on extensive databases that contain all relevant information w.r.t. availability of resources & (potential) customers. Scheduling system interacts often with forecasting & yield management modules. Fig. 1.2: Information flow diagram in a service system depicts information flow in a service organization e.g. a car rental agency. In contrast to manufacturing settings, there is usually no MRP system in a service environment.

– *Lên lịch trong Dịch vụ.* Việc mô tả 1 tổ chức dịch vụ chung & 1 hệ thống lên lịch thông thường không dễ như mô tả 1 tổ chức sản xuất chung. Chức năng lên lịch trong 1 tổ chức dịch vụ có thể gặp phải nhiều vấn đề. Nó có thể phải xử lý việc đặt trước tài nguyên, ví dụ như phân công máy bay đến cổng, hoặc đặt trước phòng họp hoặc các cơ sở khác. Các mô hình được sử dụng đôi khi hơi khác so với các mô hình được sử dụng trong môi trường sản xuất. Lên lịch trong môi trường dịch vụ phải được phối hợp với các chức năng ra quyết định khác, thường là trong các hệ thống thông tin phức tạp, tương tự như chức năng lên lịch trong môi trường sản xuất. Các hệ thống thông tin này thường dựa vào các cơ sở dữ liệu mở rộng chứa tất cả thông tin có liên quan đến tính khả dụng của tài nguyên & (khách hàng tiềm năng). Hệ thống lên lịch thường tương tác với các mô-đun dự báo & quản lý năng suất. Hình 1.2: Sơ đồ luồng thông tin trong hệ thống dịch vụ mô tả luồng thông tin trong 1 tổ chức dịch vụ, ví dụ như 1 công ty cho thuê ô tô. Trái ngược với môi trường sản xuất, thường không có hệ thống MRP trong môi trường dịch vụ.

∘ 1.3. Outline of Book. This book focuses on both theory & applications of scheduling. Theoretical side deals with detailed sequencing & scheduling of jobs. Given a collection of jobs requiring processing in a certain machine environment, problem: sequence these jobs, subject to given constraints, in such a way that 1 or more performance criteria are optimized. Scheduler may have to deal with various forms of uncertainties, e.g. random job processing times, machines subject to breakdowns, rush orders, etc.

– Cuốn sách này tập trung vào cả lý thuyết & ứng dụng của việc lập lịch. Mặt lý thuyết liên quan đến việc sắp xếp chi tiết & lập lịch các công việc. Với 1 tập hợp các công việc cần xử lý trong 1 môi trường máy móc nhất định, vấn đề: sắp xếp các công việc này, tuân theo các ràng buộc nhất định, theo cách mà 1 hoặc nhiều tiêu chí hiệu suất được tối ưu hóa. Người lập lịch có thể phải xử lý nhiều dạng bất định khác nhau, ví dụ: thời gian xử lý công việc ngẫu nhiên, máy móc dễ hỏng hóc, đơn hàng gấp, v.v.

Thousands of scheduling problems & models have been studied & analyzed in past. Obviously, only a limited number are considered in this book; selection is based on insight they provide, methodology needed for their analysis & their importance in applications.

– Hàng ngàn vấn đề lập lịch & mô hình đã được nghiên cứu & phân tích trong quá khứ. Rõ ràng, chỉ có 1 số lượng hạn chế được xem xét trong cuốn sách này; việc lựa chọn dựa trên hiểu biết mà chúng cung cấp, phương pháp cần thiết cho việc phân tích & tầm quan trọng của chúng trong các ứng dụng.

Although applications driving models in this book come mainly from manufacturing & production environments, clear from examples: scheduling plays a role in a wide variety of situations. Models & concepts considered in this book are applicable in other settings as well.

This book is divided into 3 parts. Part I (Chaps. 2–8) deals with deterministic scheduling models. In these chaps, assumed:

there are a finite number of jobs that have to be schedule with 1 or more objectives to be minimized. Emphasis is placed on analysis of relatively simple priority or dispatching rules. Chap. 2 discusses notation & gives an overview of models considered in subsequent chaps. Chaps. 3–8 consider various machine environments. Chaps. 3–4 deal with single machine, Chaps. 5 with machines in parallel, Chap. 6 with machines in series, & Chap. 7 with more complicated job shop models. Chap. 8 focuses on open shops in which there are no restrictions on routings of jobs in shop.

Part II (Chaps. 9–13) deals with stochastic scheduling models. These chaps, in most cases, also assume: a given (finite) number of jobs have to be scheduled. Job data, e.g. processing times, release dates, & due dates, may not be exactly known in advance; only their distributions are known in advance. Actual processing times, release dates, & due dates become known only at *completion* of processing or at actual occurrence of release or due date. In these models, a single objective has to be minimized, usually in expectation. Again, an emphasis is placed on analysis of relatively simple priority or dispatching rules. Chap. 9 contains preliminary material. Chap. 10 covers single machine environment. Chap. 11 also covers single machine, but in this chap, assumed: jobs are released at different points in time. This chap establishes relationship between stochastic scheduling & theory of priority queues. Chap. 12 focuses on machines in parallel & Chap. 13 describes more complicated flow shop, job shop, & open shop models.

Part III (Chaps. 14–20) deals with applications & implementation issues. Algorithms are described for a number of real-world scheduling problems. Design issues for scheduling systems are discussed & some examples of scheduling systems are given. Chaps. 14–15 describe various general purpose procedures that have proven to be useful in industrial scheduling systems. Chap. 16 describes a number of real-world scheduling problems & how they have been dealt with in practice. Chap. 17 focuses on basic issues concerning design, development, & implementation of scheduling systems, & Chap. 18 discusses more advanced concepts in design & implementation of scheduling systems. Chap. 19 gives some examples of actual implementations. Chaps. 20 ponders on what lies ahead in scheduling.

Appendices A–D present short overviews of some of basic methodologies, namely mathematical programming, dynamic programming, constraint programming, & complexity theory. Appendix E contains a complexity classification of deterministic scheduling problems, while Appendix F presents an overview of stochastic scheduling problems. Appendix G lists a number of scheduling systems that have been developed in industry & academia. Appendix H provides some guidelines for using LEKIN scheduling system which can be downloaded from author's homepage.

This book has been designed for either a master's level course or a beginning PhD level course in Production Scheduling with prerequisites being an elementary course in Operations Research & an elementary course in stochastic processes. A senior-level course can cover some of sects in Parts I & III. Such a course can be given without getting into complexity theory: one can go through chaps of Part I skipping all complexity proofs without loss of continuity. A master's level course may cover some sects in Part II as well. Even though all 3 parts are fairly self-contained, helpful to go through Chap. 2 before venturing into Part II.

– Cuốn sách này được thiết kế cho khóa học trình độ thạc sĩ hoặc khóa học trình độ tiến sĩ cơ bản về Lập lịch sản xuất với các điều kiện tiên quyết là khóa học cơ bản về Nghiên cứu hoạt động & khóa học cơ bản về quy trình ngẫu nhiên. 1 khóa học trình độ cao cấp có thể bao gồm 1 số giáo phái trong Phần I & III. 1 khóa học như vậy có thể được cung cấp mà không cần đi sâu vào lý thuyết phức tạp: người ta có thể học các chương của Phần I bỏ qua tất cả các bằng chứng về độ phức tạp mà không mất tính liên tục. 1 khóa học trình độ thạc sĩ cũng có thể bao gồm 1 số giáo phái trong Phần II. Mặc dù cả 3 phần đều khá độc lập, nhưng vẫn hữu ích khi học Chương 2 trước khi bắt đầu Phần II.

- Comments & Refs. Over last 5 decade many books have appeared that focus on sequencing & scheduling. These books range from elementary to very advanced.

  A volume edited by Muth & Thompson (1963) contains a collection of papers focusing primarily on computational aspects of scheduling. 1 of better known textbooks is the one by Conway, Maxwell, & Miller (1967) (which, even though slightly out of date, is still very interesting); this book also deals with some of stochastic aspects & with priority queues. A more recent text by Baker (1974) gives an excellent overview of many aspects of deterministic scheduling. However, this book does not deal with computational complexity issues since it appeared just before research in computational complexity started to become popular. Book by Coffman1976 is a compendium of papers on deterministic scheduling; it does cover computational complexity. An introductory textbook by French1982 covers most of techniques used in deterministic scheduling. Proceedings of a NATO workshop, edited by Dempster, Lenstra, & Rinnooy Kan (1982), contains a number of advanced papers on deterministic + on stochastic scheduling. p. 9+++

PART I: DETERMINISTIC MODELS.

- 2. Deterministic Models: Preliminaries. Over last 50 years a considerable amount of research effort has been focused on deterministic scheduling. Number & variety of models considered is astounding. During this time a notation has evolved that succinctly captures structure of many (but for sure not all) deterministic models that have been considered in literature.

  – Trong 50 năm qua, 1 lượng lớn nỗ lực nghiên cứu đã tập trung vào lập lịch xác định. Số lượng & sự đa dạng của các mô hình được xem xét thật đáng kinh ngạc. Trong thời gian này, 1 ký hiệu đã phát triển, tóm tắt ngắn gọn cấu trúc của nhiều (nhưng chắc chắn không phải tất cả) các mô hình xác định đã được xem xét trong tài liệu.

  1st sect in this chap presents an adapted version of this notation. 2nd sect contains a number of examples & describes some of shortcomings of framework & notation. 3rd sect describes several classes of schedules. A class of schedules is typically characterized by freedom scheduler has in decision-making process. Last sect discusses complexity of scheduling problems introduced in 1st sect. This last sect can be used, together with Appendices D–E, to classify scheduling problems according to their complexity.

– Phần 1 trong chương này trình bày 1 phiên bản được điều chỉnh của ký hiệu này. Phần 2 chứa 1 số ví dụ & mô tả 1 số thiếu sót của khung & ký hiệu. Phần 3 mô tả 1 số lớp lịch trình. 1 lớp lịch trình thường được đặc trưng bởi sự tự do mà bộ lập lịch có trong quá trình ra quyết định. Phần cuối cùng thảo luận về độ phức tạp của các bài toán lập lịch được giới thiệu trong phần 1. Phần cuối cùng này có thể được sử dụng, cùng với Phụ lục D–E, để phân loại các bài toán lập lịch theo độ phức tạp của chúng.

◦ 2.1. Framework & Notation. In all scheduling problems considered number of jobs & number of machines are assumed to be finite. Number of jobs is denoted by n & number of machines by m. Usually, subscript j refers to a job while subscript i refers to a machine. If a job requires a number of processing steps or operations, then pair (i, j) refers to processing step or operation of job j on machine i. Following pieces of data are associated with job j.

– Trong tất cả các bài toán lập lịch, số lượng công việc & số lượng máy được coi là hữu hạn. Số lượng công việc được ký hiệu là n & số lượng máy được ký hiệu là m. Thông thường, chỉ số j chỉ 1 công việc, trong khi chỉ số i chỉ 1 máy. Nếu 1 công việc yêu cầu 1 số bước xử lý hoặc thao tác, thì cặp (i, j) chỉ bước xử lý hoặc thao tác của công việc j trên máy i. Các dữ liệu sau được liên kết với công việc j.

1. **Processing time** p_{ij} represents processing time of job j on machine i. Subscript i is omitted if processing time of job j does not depend on machine or if job j is only to be processed on 1 given machine.
   – **Thời gian xử lý** p_{ij} biểu thị thời gian xử lý của công việc j trên máy i. Chỉ số i được bỏ qua nếu thời gian xử lý của công việc j không phụ thuộc vào máy hoặc nếu công việc j chỉ được xử lý trên 1 máy nhất định.
2. **Release date** r_j**.** Release date r_j of job j may also be referred to as ready date. It is the time the job arrives at system, i.e., earliest time at which job j can start its processing.
   – **Ngày phát hành** r_j**.** Ngày phát hành r_j của công việc j cũng có thể được gọi là ngày sẵn sàng. Đây là thời điểm công việc đến hệ thống, tức là thời điểm sớm nhất mà công việc j có thể bắt đầu xử lý.
3. **Due date** d_j**.** Due date d_j of job j represents committed shipping or completion date (i.e., data job is promised to customer). Completion of a job after its due date is allowed, but then a penalty is incurred. When a due date must be met it is referred to as a deadline & denoted by $\bar{d}_j$.
   – **Ngày đến hạn** d_j**.** Ngày đến hạn d_j của công việc j đại diện cho ngày giao hàng hoặc ngày hoàn thành đã cam kết (tức là công việc dữ liệu đã được hứa với khách hàng). Việc hoàn thành công việc sau ngày đến hạn được cho phép, nhưng sau đó sẽ bị phạt. Khi phải đáp ứng ngày đến hạn, ngày đó được gọi là hạn chót & được ký hiệu là $\bar{d}_j$.
4. **Weight** w_j**.** Weight w_j of job j is basically a priority factor, denoting importance of job j relative to the other jobs in system. E.g., this weight may represent actual cost of keeping job in system. This cost could be a holding or inventory cost; it also could represent amount of value already added to job.
   – **Trọng số** w_j**.** Trọng số w_j của công việc j về cơ bản là 1 hệ số ưu tiên, biểu thị tầm quan trọng của công việc j so với các công việc khác trong hệ thống. Ví dụ: trọng số này có thể đại diện cho chi phí thực tế để duy trì công việc trong hệ thống. Chi phí này có thể là chi phí lưu kho hoặc chi phí tồn kho; nó cũng có thể đại diện cho lượng giá trị đã được thêm vào công việc.

A scheduling problem is described by a triplet $\alpha|\beta|\gamma$. α field describes machine environment & contains just 1 entry. β field provides details of processing characteristics & constraints & may contain no entry at all, a single entry, or multiple entries. γ field describes objective to be minimized & often contains a single entry. Possible machine environments specified in α field are as follows:

– 1 vấn đề lập lịch được mô tả bằng bộ ba $\alpha|\beta|\gamma$. Trường α mô tả môi trường máy & chỉ chứa 1 mục nhập. Trường β cung cấp chi tiết về các đặc điểm xử lý & các ràng buộc & có thể không chứa mục nhập nào, chỉ chứa 1 mục nhập hoặc nhiều mục nhập. Trường γ mô tả mục tiêu cần tối thiểu hóa & thường chứa 1 mục nhập. Các môi trường máy có thể được chỉ định trong trường α như sau:

∗ **Single machine 1.** Case of a single machine is simplest of all possible machine environments & is a special case of all other more complicated machine environments.
  – **Máy đơn 1.** Trường hợp máy đơn là môi trường máy đơn giản nhất trong tất cả các môi trường máy có thể & là trường hợp đặc biệt của tất cả các môi trường máy phức tạp hơn.

∗ **Identical machines in parallel** Pm**.** There are m identical machines in parallel. Job j requires a single operation & may be processed on any 1 of m machines or on any one that belongs to a given subset. If job j cannot be processed on just any machine, but only on any one belonging to a specific subset M_j, then entry M_j appears in β field.
  – **Các máy giống hệt nhau song song** Pm**.** Có m máy giống hệt nhau song song. Công việc j yêu cầu 1 thao tác duy nhất & có thể được xử lý trên bất kỳ 1 trong m máy hoặc trên bất kỳ máy nào thuộc 1 tập hợp con nhất định. Nếu công việc j không thể được xử lý trên bất kỳ máy nào, mà chỉ trên bất kỳ máy nào thuộc 1 tập hợp con cụ thể M_j, thì mục M_j sẽ xuất hiện trong trường β.

∗ **Machines in parallel with different speeds** Qm**.** There are m machines in parallel with different speeds. Speed of machine i is denoted by v_i. Time p_{ij} that job j spends on machine i is equal to $\frac{p_j}{v_i}$ (assuming job j receives all its processing from machine i). This environment is referred to as *uniform* machines. If all machines have same speed, i.e., $v_i = 1\ \forall i$, $p_{ij} = p_j$, then environment is identical to previous one.
  – **Các máy song song với tốc độ khác nhau** Qm**.** Có m máy song song với tốc độ khác nhau. Tốc độ của máy i được ký hiệu là v_i. Thời gian p_{ij} mà công việc j dành cho máy i bằng $\frac{p_j}{v_i}$ (giả sử công việc j nhận toàn bộ xử lý từ máy i). Môi trường này được gọi là *đồng nhất* máy. Nếu tất cả các máy có cùng tốc độ, tức là $v_i = 1\ \forall i$, $p_{ij} = p_j$, thì môi trường giống hệt với môi trường trước đó.

* **Unrelated machine in parallel** Rm**.** This environment is a further generalization of previous one. There are m different machines in parallel. Machine i can process job j at speed v_{ij}. Time p_{ij} that job j spends on machine i is equal to $\frac{p_j}{v_{ij}}$ (again assuming job j receives all its processing from machine i). If speeds of machines are independent of jobs, i.e., $v_{ij} = v_i \ \forall i, j$, then environment is identical to previous one.
  – **Máy không liên quan song song** Rm**.** Môi trường này là 1 khái quát hóa thêm của môi trường trước đó. Có m máy khác nhau song song. Máy i có thể xử lý công việc j với tốc độ v_{ij}. Thời gian p_{ij} mà công việc j dành cho máy i bằng $\frac{p_j}{v_{ij}}$ (một lần nữa giả sử công việc j nhận toàn bộ xử lý từ máy i). Nếu tốc độ của các máy không phụ thuộc vào công việc, tức là $v_{ij} = v_i \ \forall i, j$, thì môi trường giống hệt với môi trường trước đó.
* **Flow shop** Fm**.** There are m machines in series. Each job has to be processed on each 1 of m machines. All jobs have to follow same route, i.e., they have to be processed 1st on machine 1, then on machine 2, & so on. After completion on 1 machine a job joins queue at next machine. Usually, all queues are assumed to operate under FIFO discipline, i.e., a job cannot "pass" another while waiting in a queue. If FIFO discipline is in effect flow shop is referred to as a *permutation* flow shop & β field includes entry *prmu*.
  – **Flow shop** Fm**.** Có m máy nối tiếp nhau. Mỗi công việc phải được xử lý trên mỗi 1 trong m máy. Tất cả các công việc phải theo cùng 1 lộ trình, tức là chúng phải được xử lý trước trên máy 1, sau đó trên máy 2, & cứ thế. Sau khi hoàn thành trên 1 máy, 1 công việc sẽ được thêm vào hàng đợi ở máy tiếp theo. Thông thường, tất cả các hàng đợi được coi là hoạt động theo nguyên tắc 1st In 1st Out FIFO, tức là 1 công việc không thể "chuyển" sang 1 công việc khác khi đang chờ trong hàng đợi. Nếu nguyên tắc FIFO có hiệu lực, flow shop được gọi là *permutation* flow shop & trường β bao gồm mục *prmu*.
* **Flexible flow shop** FFc**.** A flexible flow shop is a generalization of flow shop & parallel machine environments. Instead of m machines in series there are c stages in series with at each stage a number of identical machines in parallel. Each job has to be processed 1st at stage 1, then at stage 2, & so on. A stage functions as a bank of parallel machines; at each stage job j requires processing on only 1 machine & any machine can do. Queues between various stages may or may not operate according to 1st Come 1st Served (FCFS) discipline. (Flexible flow shops have in literature at times also been referred to as hybrid flow shops & as multiprocessor flow shops.)
  – **Xưởng xử lý dòng chảy linh hoạt** FFc**.** Xưởng xử lý dòng chảy linh hoạt là 1 khái quát hóa của xưởng xử lý dòng chảy & môi trường máy song song. Thay vì m máy nối tiếp, sẽ có c giai đoạn nối tiếp, với mỗi giai đoạn là 1 số máy giống hệt nhau song song. Mỗi tác vụ phải được xử lý trước ở giai đoạn 1, sau đó ở giai đoạn 2, & cứ thế. 1 giai đoạn hoạt động như 1 ngân hàng các máy song song; tại mỗi giai đoạn, tác vụ j chỉ yêu cầu xử lý trên 1 máy & bất kỳ máy nào cũng có thể thực hiện. Hàng đợi giữa các giai đoạn khác nhau có thể hoặc không hoạt động theo nguyên tắc "Ai đến trước được phục vụ trước" (FCFS). (Trong tài liệu, xưởng xử lý dòng chảy linh hoạt đôi khi còn được gọi là xưởng xử lý dòng chảy lai & xưởng xử lý dòng chảy đa bộ xử lý.)
* **Job shop** Jm**.** In a job shop with m machines each job has its own predetermined route to follow. A distinction is made between job shops in which each job visits each machine at most once & job shops in which a job may visit each machine more than once. In latter case β-field contains entry *rcrc* for *recirculation*.
  – **Xưởng gia công** Jm**.** Trong 1 xưởng gia công với m máy, mỗi công việc có 1 lộ trình riêng được xác định trước để thực hiện. Có sự phân biệt giữa xưởng gia công mà mỗi công việc đến mỗi máy tối đa 1 lần & xưởng gia công mà 1 công việc có thể đến mỗi máy nhiều hơn 1 lần. Trong trường hợp sau, trường β chứa mục *rcrc* cho *tuần hoàn*.
* **Flexible job shop** FJc**.** A flexible job shop is a generalization of job shop & parallel machine environments. Instead of m machines in series there are c workcenters with at each workcenter a number of identical machines in parallel. Each job has its own route to follow through shop; job j requires processing at each workcenter on only 1 machine & any machine can do. If a job on its route through shop may visit a workcenter more than once, then β-field contains entry *rcrc* for recirculation.
  – **Xưởng gia công linh hoạt** FJc**.** Xưởng gia công linh hoạt là 1 khái quát hóa của môi trường xưởng gia công & máy song song. Thay vì m máy nối tiếp, sẽ có c trung tâm gia công, với mỗi trung tâm gia công là 1 số máy giống hệt nhau được bố trí song song. Mỗi công việc có 1 lộ trình riêng để đi qua xưởng; công việc j yêu cầu xử lý tại mỗi trung tâm gia công chỉ trên 1 máy & bất kỳ máy nào cũng có thể thực hiện. Nếu 1 công việc trên lộ trình qua xưởng có thể ghé thăm trung tâm gia công nhiều hơn 1 lần, thì trường β chứa mục *rcrc* để tuần hoàn.
* **Open shop** Om**.** There are m machines. Each job has to be processed again on each 1 of m machines. However, some of these processing times may be 0. There are no restrictions with regard to routing of each job through machine environment. Scheduler is allowed to determine a route for each job & different jobs may have different routes.
  – **Mở cửa hàng** Om**.** Có m máy. Mỗi công việc phải được xử lý lại trên mỗi 1 trong m máy. Tuy nhiên, 1 số thời gian xử lý này có thể bằng 0. Không có hạn chế nào về việc định tuyến từng công việc qua môi trường máy. Bộ lập lịch được phép xác định tuyến đường cho mỗi công việc & các công việc khác nhau có thể có các tuyến đường khác nhau.

Processing restrictions & constraints specified in β field may include multiple entries. Possible entries in β field are as follows:
– Các hạn chế xử lý & ràng buộc được chỉ định trong trường β có thể bao gồm nhiều mục nhập. Các mục nhập có thể có trong trường β như sau:

* **Release dates** r_j**.** if this symbol appears in β field, then job j cannot start its processing before its release date r_j. If r_j does not appear in β field, processing of job j may start at any time. In contrast to release dates, due dates are not specified in this field. type of objective function gives sufficient indication whether or not there are due dates.
  – **Ngày phát hành** r_j**.** nếu ký hiệu này xuất hiện trong trường β, thì công việc j không thể bắt đầu xử lý trước ngày phát hành r_j. Nếu r_j không xuất hiện trong trường β, việc xử lý công việc j có thể bắt đầu bất cứ lúc nào. Trái ngược

với ngày phát hành, ngày đến hạn không được chỉ định trong trường này. Kiểu hàm mục tiêu cung cấp đủ thông tin về việc có ngày đến hạn hay không.

* **Preemptions** *prmp*. Preemptions imply that not necessary to keep a job on a machine, once started, until its completion. Scheduler is allowed to interrupt processing of a job (preempt) any any point in time & put a different job on machine instead. Amount of processing a preempted job already has received is not lost. When a preempted job is afterward put back on machine (or on another machine in case of parallel machines), it only needs machine for its *remaining* processing time. When preemptions are allowed *prmp* is included in β field; when *prmp* is not included, preemptions are not allowed.
  – **Quyền ưu tiên** *prmp*. Quyền ưu tiên ngụ ý rằng không cần thiết phải giữ 1 công việc trên máy, 1 khi đã bắt đầu, cho đến khi hoàn thành. Bộ lập lịch được phép ngắt quá trình xử lý 1 công việc (quyền ưu tiên) tại bất kỳ thời điểm nào & đặt 1 công việc khác vào máy thay thế. Lượng xử lý mà 1 công việc đã bị chiếm dụng trước đó nhận được sẽ không bị mất. Khi 1 công việc đã bị chiếm dụng sau đó được đặt lại trên máy (hoặc trên 1 máy khác trong trường hợp máy song song), nó chỉ cần máy trong thời gian xử lý *còn lại* của nó. Khi được phép chiếm dụng, *prmp* được bao gồm trong trường β; khi *prmp* không được bao gồm, quyền ưu tiên không được phép.
* **Precedence constraints** *prec*. Precedence constraints may appear in a single machine or in a parallel machine environment, requiring that 1 or more jobs may have to be completed before another job is allowed to start its processing. There are several special forms of precedence constraints: if each job has at most 1 predecessor & at most 1 successor, constraints are referred to as *chains*. If each job has at most 1 successor, constraints are referred to as an *intree*. If each job has at most 1 predecessor, constraints are referred to as an *outtree*. If no *prec* appears in β field, jobs are not subject to precedence constraints.
  – **Ràng buộc thứ tự ưu tiên** *prec*. Ràng buộc thứ tự ưu tiên có thể xuất hiện trong 1 máy đơn hoặc trong môi trường máy song song, yêu cầu 1 hoặc nhiều công việc phải được hoàn thành trước khi 1 công việc khác được phép bắt đầu xử lý. Có 1 số dạng ràng buộc thứ tự ưu tiên đặc biệt: nếu mỗi công việc có nhiều nhất 1 công việc tiền nhiệm & nhiều nhất 1 công việc kế nhiệm, các ràng buộc được gọi là *chuỗi*. Nếu mỗi công việc có nhiều nhất 1 công việc kế nhiệm, các ràng buộc được gọi là *cây nội bộ*. Nếu mỗi công việc có nhiều nhất 1 công việc tiền nhiệm, các ràng buộc được gọi là *cây ngoại vi*. Nếu không có *prec* nào xuất hiện trong trường β, các công việc không phải tuân theo ràng buộc thứ tự ưu tiên.
* **Sequence dependent setup times** s_{jk}. s_{jk} represents sequence dependent setup time that is incurred between processing of jobs j, k; s_{0k} denotes setup time for job k if job k is 1st in sequence & s_{j0} clean-up time after job j if job j is last in sequence (of course, s_{0k}, s_{j0} may be 0). If setup time between jobs j, k depends on machine, then subscript i is included, i.e., s_{ijk}. If no s_{jk} appears in β field, all setup times are assumed to be 0 or sequence independent, in which case they are simply included in processing times.
  – **Thời gian thiết lập phụ thuộc vào trình tự** s_{jk}. s_{jk} biểu thị thời gian thiết lập phụ thuộc vào trình tự phát sinh giữa các lần xử lý công việc j, k; s_{0k} biểu thị thời gian thiết lập cho công việc k nếu công việc k là công việc đầu tiên trong trình tự & s_{j0} thời gian dọn dẹp sau công việc j nếu công việc j là công việc cuối cùng trong trình tự (tất nhiên, s_{0k}, s_{j0} có thể bằng 0). Nếu thời gian thiết lập giữa các công việc j, k phụ thuộc vào máy, thì chỉ số dưới i được bao gồm, tức là s_{ijk}. Nếu không có s_{jk} nào xuất hiện trong trường β, thì tất cả các thời gian thiết lập được coi là bằng 0 hoặc không phụ thuộc vào trình tự, trong trường hợp đó, chúng chỉ đơn giản được bao gồm trong thời gian xử lý.
* **Job families** $fmls$. n jobs belong in this case to F different job families. Jobs from same family may have different processing times, but they can be processed on a machine one after another without requiring any setup in between. However, if machine switches over from 1 family to another, say from family g to family h, then a setup is required. If this setup time depends on both families g, h & is sequence dependent, then it is denoted by s_{gh}. If this setup time depends only on family about to start, i.e., family h, then it is denoted by s_h. If it does not depend on either family, it is denoted by s.
  – **Họ công việc** $fmls$. n công việc trong trường hợp này thuộc về F họ công việc khác nhau. Các công việc từ cùng 1 họ có thể có thời gian xử lý khác nhau, nhưng chúng có thể được xử lý trên 1 máy lần lượt mà không cần bất kỳ thiết lập nào ở giữa. Tuy nhiên, nếu máy chuyển từ họ này sang họ khác, chẳng hạn từ họ g sang họ h, thì cần phải thiết lập. Nếu thời gian thiết lập này phụ thuộc vào cả hai họ g, h & phụ thuộc vào trình tự, thì nó được ký hiệu là s_{gh}. Nếu thời gian thiết lập này chỉ phụ thuộc vào họ sắp bắt đầu, tức là họ h, thì nó được ký hiệu là s_h. Nếu nó không phụ thuộc vào bất kỳ họ nào, thì nó được ký hiệu là s.
* **Batching processing** $batch(b)$. A machine may be able to process a number of jobs, say b, simultaneously; i.e., it can process a batch of up to b jobs at same time. Processing times of jobs in a batch may not be all same & entire batch is finished only when last job of batch has been completed, implying that completion time of entire batch is determined by job with longest processing time. If $b = 1$, then problem reduces to a conventional scheduling environment. Another special case of interest is $b = \infty$, i.e., there is no limit on the number of jobs machine can handle at any time.
  – **Xử lý theo lô** $batch(b)$. 1 máy có thể xử lý đồng thời 1 số tác vụ, chẳng hạn b; nghĩa là, nó có thể xử lý 1 lô tối đa b tác vụ cùng 1 lúc. Thời gian xử lý của các tác vụ trong 1 lô có thể không giống nhau & toàn bộ lô chỉ được hoàn thành khi tác vụ cuối cùng của lô đã hoàn thành, ngụ ý rằng thời gian hoàn thành của toàn bộ lô được xác định bởi tác vụ có thời gian xử lý dài nhất. Nếu $b = 1$, thì bài toán được rút gọn về môi trường lập lịch thông thường. 1 trường hợp đặc biệt khác đáng quan tâm là $b = \infty$, nghĩa là không có giới hạn về số lượng tác vụ mà máy có thể xử lý tại bất kỳ thời điểm nào.
* **Breakdowns** *brkdwn*. Machine breakdowns imply that a machine may not be continuously available. Periods that a machine is not available are, in this part of book, assumed to be fixed (e.g., due to shifts or scheduled maintenance). If there are a number of identical machines in parallel, number of machines available at any point in time is a function of time, i.e., $m(t)$. Machine breakdowns are at times also referred to as machine availability constraints.

– **Hỏng hóc** *brkdwn*. Hỏng hóc máy móc ngụ ý rằng máy móc có thể không hoạt động liên tục. Trong phần này của sách, những khoảng thời gian máy móc không hoạt động được coi là cố định (ví dụ: do ca làm việc hoặc bảo trì theo lịch trình). Nếu có nhiều máy móc giống hệt nhau hoạt động song song, số lượng máy móc khả dụng tại bất kỳ thời điểm nào là 1 hàm số theo thời gian, tức là $m(t)$. Hỏng hóc máy móc đôi khi còn được gọi là ràng buộc về khả năng hoạt động của máy móc.

* **Machine eligibility restrictions** M_j. M_j symbol may appear in β field when machine environment is m machines in parallel Pm. When M_j is present, not all m machines are capable of processing job j. Set M_j denotes set of machines that can process job j. If β field does not contain M_j, job j may be processed on any 1 of m machines.
– **Giới hạn điều kiện máy** M_j. Ký hiệu M_j có thể xuất hiện trong trường β khi môi trường máy là m máy song song Pm. Khi có M_j, không phải tất cả m máy đều có khả năng xử lý công việc j. Tập M_j biểu thị tập hợp các máy có thể xử lý công việc j. Nếu trường β không chứa M_j, công việc j có thể được xử lý trên bất kỳ 1 trong m máy.
* **Permutation** *prmu*. A constraint that may appear in flow shop environment: queues in front of each machine operate according to FIFO discipline. This implies: order (or *permutation*) in which jobs go through 1st machine is maintained throughout system.
– **Hoán vị** *prmu*. 1 ràng buộc có thể xuất hiện trong môi trường flow shop: hàng đợi trước mỗi máy hoạt động theo nguyên tắc FIFO. Điều này ngụ ý: thứ tự (hoặc *hoán vị*) mà các công việc đi qua máy thứ nhất được duy trì trong toàn bộ hệ thống.
* **Blocking** *block*. Blocking is a phenomenon that may occur in flow shops. If a flow shop has a limited buffer in between 2 successive machines, then it may happen that when buffer is full upstream machine is not allowed to release a completed job. Blocking implies: completed job has to remain on upstream machine preventing (i.e., blocking) that machine from working on next job. Most common occurrence of blocking considered in this book: case with zero buffers in between any 2 successive machines. In this case a job that has completed its processing on a given machine cannot leave machine if preceding job has not yet completed its processing on next machine; thus, blocked job also prevents (or blocks) next job from starting its processing on given machine. In models with blocking that are considered in subsequent chaps, assumption is made that machines operate according to FIFO. I.e., *block* implies *prmu*.
– **Chặn** *khi*. Chặn là 1 hiện tượng có thể xảy ra trong các xưởng sản xuất dòng chảy. Nếu 1 xưởng sản xuất dòng chảy có bộ đệm giới hạn giữa 2 máy liên tiếp, thì có thể xảy ra trường hợp khi bộ đệm đầy, máy thượng nguồn không được phép giải phóng 1 công việc đã hoàn thành. Chặn ngụ ý: công việc đã hoàn thành phải nằm trên máy thượng nguồn, ngăn (tức là chặn) máy đó làm việc trên công việc tiếp theo. Trường hợp chặn phổ biến nhất được xem xét trong cuốn sách này: trường hợp có bộ đệm bằng không giữa bất kỳ 2 máy liên tiếp nào. Trong trường hợp này, 1 công việc đã hoàn thành quá trình xử lý trên 1 máy nhất định không thể rời khỏi máy nếu công việc trước đó vẫn chưa hoàn thành quá trình xử lý trên máy tiếp theo; do đó, công việc bị chặn cũng ngăn (hoặc chặn) công việc tiếp theo bắt đầu quá trình xử lý của nó trên máy nhất định. Trong các mô hình có chặn được xem xét trong các chương tiếp theo, giả định được đưa ra là các máy hoạt động theo FIFO. Tức là, *khi* ngụ ý *prmu*.
* **no-wait** *nwt*. *no-wait* requirement is another phenomenon that may occur in flow shops. Jobs are not allowed to wait between 2 successive machines. This implies: starting time of a job at 1st machine has to be delayed to ensure: job can go through flow shop without having to wait for any machine. An example of such an operation is a steel rolling mill in which a slab of steel is not allowed to wait as it would cool off during a wait. Clear: under no-wait machines also operate according to FIFO discipline.
– Yêu cầu **no-wait** *nwt*. *no-wait* là 1 hiện tượng khác có thể xảy ra trong các xưởng sản xuất. Các công việc không được phép chờ giữa 2 máy liên tiếp. Điều này ngụ ý: thời gian bắt đầu của 1 công việc tại máy đầu tiên phải được trì hoãn để đảm bảo: công việc có thể đi qua xưởng sản xuất mà không phải chờ bất kỳ máy nào. 1 ví dụ về hoạt động như vậy là 1 nhà máy cán thép, trong đó 1 tấm thép không được phép chờ vì nó sẽ nguội đi trong thời gian chờ. Rõ ràng: trong các máy không chờ, các máy cũng hoạt động theo nguyên tắc FIFO.
* **Recirculation** *rcrc*. Recirculation may occur in a job shop or flexible job shop when a job may visit a machine or workcenter more than once.
– **Tuần hoàn** *rcrc*. Tuần hoàn có thể xảy ra trong xưởng gia công hoặc xưởng gia công linh hoạt khi 1 công việc có thể đến 1 máy hoặc trung tâm gia công nhiều hơn 1 lần.

Any other entry that may appear in β field is self explanatory. E.g., $p_j = p$ implies that all processing times are equal & $d_j = d$ implies that all due dates are equal. As stated before, due dates, in contrast to release dates, are usually not explicitly specified in this field; type of objective function gives sufficient indication whether or not jobs have due dates.

– Bất kỳ mục nhập nào khác có thể xuất hiện trong trường β đều tự giải thích. Ví dụ: $p_j = p$ ngụ ý rằng tất cả thời gian xử lý đều bằng nhau & $d_j = d$ ngụ ý rằng tất cả các ngày đến hạn đều bằng nhau. Như đã đề cập trước đó, ngày đến hạn, trái ngược với ngày phát hành, thường không được chỉ định rõ ràng trong trường này; loại hàm mục tiêu cung cấp đủ thông tin về việc công việc có ngày đến hạn hay không.

Objective to be minimized is always a function of completion times of jobs, which, of course, depend on schedule. Completion time of operation of job j on machine i is denoted by C_{ij}. Time job j exits system (i.e., its completion time on last machine on which it requires processing) is denoted by C_j. Objective may also be a function of due dates. *Lateness* of job j is defined as $L_j := C_j - d_j$, which is positive when job j is completed late & negative when it is completed early. *Tardiness* of job j is defined as

$$T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}.$$

Difference between tardiness & lateness lies in fact that tardiness never is negative. *Unit penalty* of job j is defined as

$$U_j = \begin{cases} 1 & \text{if } C_j > d_j, \\ 0 & \text{otherwise.} \end{cases}$$

Lateness, tardiness, & unit penalty are 3 basic due date related penalty functions considered in this book. Shape of these functions is depicted in Fig. 2.1: Due date related penalty functions.

– Mục tiêu cần tối thiểu hóa luôn là 1 hàm số của thời gian hoàn thành công việc, & tất nhiên, điều này phụ thuộc vào lịch trình. Thời gian hoàn thành hoạt động của công việc j trên máy i được ký hiệu là C_{ij}. Thời gian công việc j thoát khỏi hệ thống (tức là thời gian hoàn thành của nó trên máy cuối cùng mà nó cần xử lý) được ký hiệu là C_j. Mục tiêu cũng có thể là 1 hàm số của ngày đến hạn. *Độ trễ* của công việc j được định nghĩa là $L_j := C_j - d_j$, giá trị dương khi công việc j hoàn thành muộn & âm khi hoàn thành sớm. *Độ trễ* của công việc j được định nghĩa là

$$T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}.$$

Sự khác biệt giữa độ trễ & độ trễ nằm ở chỗ độ trễ không bao giờ là số âm. *Đơn vị phạt* của công việc j được định nghĩa là

$$U_j = \begin{cases} 1 & \text{nếu } C_j > d_j, \\ 0 & \text{nếu không.} \end{cases}$$

Độ trễ, sự chậm trễ, & đơn vị phạt là 3 hàm phạt cơ bản liên quan đến ngày đến hạn được xem xét trong cuốn sách này. Hình dạng của các hàm này được mô tả trong Hình 2.1: Hàm phạt liên quan đến ngày đến hạn.

Examples of possible objective functions to be minimized are as follows:

* **Makespan** $C_{\max}$**.** Makespan, defined as $\max\{C_1, \ldots, C_n\}$, is equivalent to completion time of last job to leave system. A minimum makespan usually implies a good utilization of machine(s).
  – **Makespan** $C_{\max}$**.** Makespan, được định nghĩa là $\max\{C_1, \ldots, C_n\}$, tương đương với thời gian hoàn thành công việc cuối cùng để rời khỏi hệ thống. Makespan tối thiểu thường ngụ ý việc sử dụng máy móc hiệu quả.
* **Maximum Lateness** $L_{\max}$**.** Maximum lateness $L_{\max}$ is defined as $\max\{L_1, \ldots, L_n\}$. It measures worst violation of due dates.
  – **Độ trễ tối đa** $L_{\max}$**.** Độ trễ tối đa $L_{\max}$ được định nghĩa là $\max\{L_1, \ldots, L_n\}$. Chỉ số này đo lường mức độ vi phạm hạn chót nghiêm trọng nhất.
* **Total weighted completion time** $\sum w_j C_j$**.** Sum of weighted completion times of n jobs gives an indication of total holding or inventory costs incurred by schedule. Sum of completion times is in literature often referred to as flow time. Total weighted completion time is then referred to as weighted flow time.
  – **Tổng thời gian hoàn thành có trọng số** $\sum w_j C_j$**.** Tổng thời gian hoàn thành có trọng số của n công việc cho biết tổng chi phí lưu kho hoặc tồn kho phát sinh theo tiến độ. Tổng thời gian hoàn thành trong tài liệu thường được gọi là thời gian lưu chuyển. Tổng thời gian hoàn thành có trọng số khi đó được gọi là thời gian lưu chuyển có trọng số.
* **Discounted total weighted completion time** $\sum w_j(1-e^{-rC_j})$**.** This is a more general cost function than the previous one, where costs are discounted at a rate of $r \in (0, 1)$, per unit time. I.e., if job j is not completed by time t an additional cost $w_j r e^{-rt} dt$ is incurred over the period $[t, t + dt]$. If job j is completed at time t, total cost incurred over period $[0, t]$ is $w_j(1 - e^{-rt})$. Value of r is usually close to 0, say 0.1 or 10%.
  – **Tổng thời gian hoàn thành có trọng số chiết khấu** $\sum w_j(1 - e^{-rC_j})$**.** Đây là hàm chi phí tổng quát hơn hàm trước, trong đó chi phí được chiết khấu theo tỷ lệ $r \in (0, 1)$, trên 1 đơn vị thời gian. Tức là, nếu công việc j không được hoàn thành vào thời điểm t thì sẽ phát sinh thêm chi phí $w_j r e^{-rt} dt$ trong khoảng thời gian $[t, t + dt]$. Nếu công việc j được hoàn thành vào thời điểm t, tổng chi phí phát sinh trong khoảng thời gian $[0, t]$ là $w_j(1-e^{-rt})$. Giá trị của r thường gần bằng 0, chẳng hạn như 0,1 hoặc 10
* **Total weighted tardiness** $\sum w_j T_j$**.** This is also a more general cost function than total weighted completion time.
  – **Tổng thời gian trễ có trọng số** $\sum w_j T_j$**.** Đây cũng là hàm chi phí tổng quát hơn so với tổng thời gian hoàn thành có trọng số.
* **Weighted number of tardy jobs** $\sum w_j U_j$**.** Weighted number of tardy jobs is not only a measure of academic interest, it is often an objective in practice as it is a measure that can be recorded very easily.
  – **Số lượng công việc đi muộn có trọng số** $\sum w_j U_j$**.** Số lượng công việc đi muộn có trọng số không chỉ là thước đo mức độ quan tâm về mặt học thuật mà còn thường là mục tiêu trong thực tế vì đây là thước đo có thể ghi lại rất dễ dàng.

All objective functions above are so-called *regular* performance measures. A regular performance measure is a function that is nondecreasing in $C_1, \ldots, C_n$. Recently researchers have begun to study objective functions that are not regular. E.g., when job j has a due date d_j, it may be subject to an earliness penalty, where *earliness* of job j is defined as $E_j = \max\{d_j - C_j, 0\}$. This earliness penalty is nonincreasing in C_j. An objective e.g. total earliness plus total tardiness, i.e.

$$\sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j,$$

is therefore not regular. A more general objective that is not regular is total weighted earliness plus total weighted tardiness, i.e.,

$$\sum_{j=1}^{n} w'_j E_j + \sum_{j=1}^{n} w''_j T_j.$$

Weight associated with earliness of job j, w'_j may be different from weight associated with tardiness of job j, w''_j.

– Tất cả các hàm mục tiêu trên đều được gọi là các thước đo hiệu suất *regular*. 1 thước đo hiệu suất regular là 1 hàm không giảm trong $C_1, \ldots, C_n$. Gần đây, các nhà nghiên cứu đã bắt đầu nghiên cứu các hàm mục tiêu không chính quy. Ví dụ: khi công việc j có ngày đến hạn là d_j, nó có thể bị phạt vì hoàn thành sớm, trong đó *earlyliness* của công việc j được định nghĩa là $E_j = \max\{d_j - C_j, 0\}$. Phạt vì hoàn thành sớm này không tăng trong C_j. 1 mục tiêu, ví dụ: tổng thời gian hoàn thành sớm cộng với tổng thời gian hoàn thành muộn, tức là

$$\sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j,$$

do đó không chính quy. 1 mục tiêu tổng quát hơn không phải là mục tiêu chính quy là tổng thời gian đi sớm có trọng số cộng với tổng thời gian đi muộn có trọng số, tức là,

$$\sum_{j=1}^{n} w'_j E_j + \sum_{j=1}^{n} w''_j T_j.$$

Trọng số liên quan đến thời gian đi sớm của công việc j, w'_j có thể khác với trọng số liên quan đến thời gian đi muộn của công việc j, w''_j.

- 2.2. Examples. The following examples illustrate the notation:

Example 5 (A flexible flow shop). *$FFc|r_j|\sum w_jT_j$ denotes a flexible flow shop. Jobs have release dates & due dates & objective is minimization of total weighted tardiness. Example 1.1.1 in Section 1.1 (the paper bag factory) can be modeled as such. Actually, problem described in Sect. 1.1 has some additional characteristics including sequence dependent setup times at each of 3 stages. In addition, processing time of job j on machine i has a special structure: it depends on number of bags & on speed of machine.*

– $FFc|r_j|\sum w_jT_j$ biểu thị 1 xưởng sản xuất linh hoạt. Các công việc có ngày phát hành & ngày đến hạn & mục tiêu là giảm thiểu tổng số lần chậm trễ có trọng số. Ví dụ 1.1.1 trong Mục 1.1 (nhà máy sản xuất túi giấy) có thể được mô hình hóa như vậy. Thực tế, bài toán được mô tả trong Mục 1.1 có 1 số đặc điểm bổ sung bao gồm thời gian thiết lập phụ thuộc vào trình tự tại mỗi giai đoạn trong 3 giai đoạn. Ngoài ra, thời gian xử lý của công việc j trên máy i có 1 cấu trúc đặc biệt: phụ thuộc vào số lượng túi & tốc độ của máy.

Example 6 (A flexible job shop). *$FJc|r_j, s_{ijk}, rcrc|\sum w_jT_j$ refers to a flexible job shop with c workcenters. Jobs have different release dates & are subject to sequence dependent setup times that are machine dependent. There is recirculation, so a job may visit a work center more than once. Objective: minimize total weighted tardiness. Clear: this this problem is a more general problem than the one described in previous example. Example 1.1.2 in Sect. 1.1 (emiconductor manufacturing facility) can be modeled as such.*

– $FJc|r_j, s_{ijk}, rcrc|\sum w_jT_j$ đề cập đến 1 xưởng gia công linh hoạt với c trung tâm gia công. Các công việc có ngày phát hành khác nhau & phải tuân theo thời gian thiết lập phụ thuộc vào trình tự & phụ thuộc vào máy. Có sự tuần hoàn, vì vậy 1 công việc có thể đến trung tâm gia công nhiều lần. Mục tiêu: giảm thiểu tổng số lần trễ có trọng số. Rõ ràng: đây là 1 vấn đề tổng quát hơn so với vấn đề được mô tả trong ví dụ trước. Ví dụ 1.1.2 trong Mục 1.1 (nhà máy sản xuất chất bán dẫn) có thể được mô hình hóa như vậy.

Example 7 (A parallel machine environment). *$Pm|r_j, M_j|\sum w_jT_j$ denotes a system with m machines in parallel. Job j arrives at release data r_j & has to leave by due date d_j. Job j may be processed only on 1 of machines belonging to subset M_j. If job j is not completed in time a penalty w_jT_j is incurred. This model can be used for gate assignment problem described in Example 1.1.3.*

– $Pm|r_j, M_j|\sum w_jT_j$ biểu thị 1 hệ thống với m máy song song. Công việc j đến lúc dữ liệu phát hành r_j & phải rời đi trước ngày đến hạn d_j. Công việc j chỉ có thể được xử lý trên 1 trong các máy thuộc tập con M_j. Nếu công việc j không hoàn thành đúng hạn, sẽ bị phạt w_jT_j. Mô hình này có thể được sử dụng cho bài toán gán cổng được mô tả trong Ví dụ 1.1.3.

Example 8 (A single machine environment). *$1|r_j, prmp|\sum w_jC_j$ denotes a single machine system with job j entering system at its release date r_j. Preemptions are allowed. Objective to be minimized is sum of weighted completion times. This model can be used to study deterministic counterpart of problem described in Example 1.1.4.*

– $1|r_j, prmp|\sum w_jC_j$ biểu thị 1 hệ thống máy đơn với tác vụ j được đưa vào hệ thống vào ngày phát hành r_j. Cho phép chiếm dụng ưu tiên. Mục tiêu cần giảm thiểu là tổng thời gian hoàn thành có trọng số. Mô hình này có thể được sử dụng để nghiên cứu bài toán xác định tương ứng được mô tả trong Ví dụ 1.1.4.

Example 9 (Sequence dependent setup times). *$1|s_{jk}|C_{\max}$ denotes a single machine system with n jobs subject to sequence dependent setup times, where objective: minimize makespan. Well-known: this problem is equivalent to so-called TSP, where a salesman has to tour n cities in such a way that total distance traveled is minimized (see Appendix D for a formal definition of TSP).*

– $1|s_{jk}|C_{\max}$ biểu thị 1 hệ thống máy đơn với n công việc phụ thuộc vào thời gian thiết lập phụ thuộc vào trình tự, trong đó mục tiêu: giảm thiểu thời gian chờ. Bài toán này tương đương với bài toán TSP, trong đó 1 nhân viên bán hàng phải đi qua n thành phố sao cho tổng quãng đường di chuyển là nhỏ nhất (xem Phụ lục D để biết định nghĩa chính thức của TSP).

Example 10 (A project). *$P\infty|prec|C_{\max}$ denotes a scheduling problem with n jobs subject to precedence constraints & an unlimited number of machines (or resources) in parallel. Total time of entire project has to be minimized. This type of problem is very common in project planning in construction industry & has led to techniques e.g. Critical Path Method (CPM) & Project Evaluation & Review Technique (PERT).*

– $P\infty|prec|C_{\max}$ biểu thị 1 bài toán lập lịch với n công việc chịu ràng buộc về thứ tự ưu tiên & số lượng máy móc (hoặc tài nguyên) không giới hạn hoạt động song song. Tổng thời gian của toàn bộ dự án phải được giảm thiểu. Loại bài toán này rất phổ biến trong lập kế hoạch dự án trong ngành xây dựng & đã dẫn đến các kỹ thuật như Phương pháp Đường găng (CPM) & Kỹ thuật Đánh giá Dự án & Kỹ thuật Rà soát (PERT).

Example 11 (A flow shop). *$Fm|p_{ij} = p_j|\sum w_j C_j$ denotes a proportionate flow shop environment with m machines in series; processing times of job j on all m machines are identical & equal to p_j (hence term proportionate). Objective: find order in which n jobs go through system so that sum of weighted completion times is minimized.*

– $Fm|p_{ij} = p_j|\sum w_j C_j$ biểu thị 1 môi trường sản xuất theo quy trình tỷ lệ với m máy nối tiếp; thời gian xử lý công việc j trên tất cả m máy đều giống hệt nhau & bằng p_j (do đó có thuật ngữ tỷ lệ). Mục tiêu: tìm thứ tự mà n công việc đi qua hệ thống sao cho tổng thời gian hoàn thành có trọng số là nhỏ nhất.

Example 12 (A job shop). *$Jm||C_{\max}$ denotes a job shop problem with m machines. There is no recirculation, so a job visits each machine at most once. Objective: minimize makespan. This problem is considered a classic in scheduling literature & has received an enormous amount of attention.*

– $Jm||C_{\max}$ biểu thị 1 bài toán xưởng gia công với m máy. Không có sự tuần hoàn, vì vậy 1 công việc chỉ đến được mỗi máy tối đa 1 lần. Mục tiêu: giảm thiểu thời gian chờ. Bài toán này được coi là 1 bài toán kinh điển trong các tài liệu lập lịch trình & đã nhận được rất nhiều sự quan tâm.

Of course, there are many scheduling models that are not captured by this framework. One can define, e.g., a more general flexible job shop in which each workcenter consists of a number of unrelated machines in parallel. When a job on its route through system arrives at a bank of unrelated machines, it may be processed on any 1 of machines, but its processing time now depends on machine on which it is processed.

– Tất nhiên, có nhiều mô hình lập lịch không được khung này nắm bắt. Ví dụ, ta có thể định nghĩa 1 xưởng làm việc linh hoạt tổng quát hơn, trong đó mỗi trung tâm làm việc bao gồm 1 số máy không liên quan hoạt động song song. Khi 1 công việc trên đường đi qua hệ thống đến 1 nhóm máy không liên quan, nó có thể được xử lý trên bất kỳ máy nào, nhưng thời gian xử lý của nó phụ thuộc vào máy mà nó được xử lý.

One can also define a model that is a mixture of a job shop & an open shop. Routes of some jobs are fixed, while routes of other jobs are (partially) open.

– Người ta cũng có thể định nghĩa 1 mô hình kết hợp giữa xưởng gia công & xưởng mở. Tuyến đường của 1 số công việc là cố định, trong khi tuyến đường của các công việc khác là mở (một phần).

Framework described in Sect. 2.1 has been designed primarily for models with a single objective. Most research in past has concentrated on models with a single objective. Recently, researchers have begun studying models with multiple objectives as well.

– Khung được mô tả trong Mục 2.1 được thiết kế chủ yếu cho các mô hình có 1 mục tiêu duy nhất. Hầu hết các nghiên cứu trước đây đều tập trung vào các mô hình có 1 mục tiêu duy nhất. Gần đây, các nhà nghiên cứu cũng đã bắt đầu nghiên cứu các mô hình có nhiều mục tiêu.

Various other scheduling features, not mentioned here, have been studied & analyzed in literature. Such features include periodic or cyclic scheduling, personnel scheduling, & resource constrained scheduling.

– Nhiều tính năng lập lịch khác, không được đề cập ở đây, đã được nghiên cứu & phân tích trong các tài liệu. Các tính năng này bao gồm lập lịch định kỳ hoặc theo chu kỳ, lập lịch nhân sự, & lập lịch hạn chế nguồn lực.

- 2.3. Classes of Schedules. In scheduling terminology a distinction is often made between a *sequence*, a *schedule*, & a *scheduling policy*. A sequence usually corresponds to a permutation of n jobs or order in which jobs are to be processed on a given machine. A schedule usually refers to an allocation of jobs within a more complicated setting of machines, allowing possibly for preemptions of jobs by other jobs that are released at later points in time. Concept of a scheduling policy is often used in stochastic settings: a policy prescribes an appropriate action for any 1 of states system may be in. In deterministic models usually only sequences or schedules are of importance.

  – Các Lớp Lịch Trình. Trong thuật ngữ lập lịch, người ta thường phân biệt giữa *sequence*, *schedule* & *scheduling policy*. 1 chuỗi thường tương ứng với 1 hoán vị của n công việc hoặc thứ tự mà các công việc sẽ được xử lý trên 1 máy nhất định. 1 lịch trình thường đề cập đến việc phân bổ các công việc trong 1 thiết lập máy phức tạp hơn, cho phép các công việc khác được giải phóng tại các thời điểm sau đó có thể chiếm dụng công việc trước. Khái niệm về chính sách lập lịch thường được sử dụng trong các thiết lập ngẫu nhiên: 1 chính sách quy định 1 hành động phù hợp cho bất kỳ 1 trong các trạng thái mà hệ thống có thể ở. Trong các mô hình xác định, thường chỉ có các trình tự hoặc lịch trình là quan trọng.

  Assumptions have to be made with regard to what scheduler may & may not do when he generates a schedule. E.g., it may be case that a schedule may not have any *unforced idleness* on any machine. This class of schedules can be defined as follows.

  – Cần phải đưa ra các giả định về những gì trình lập lịch có thể & không thể làm khi tạo lịch. Ví dụ, có thể 1 lịch trình không có bất kỳ trạng thái nhàn rỗi tự phát nào trên bất kỳ máy nào. Lớp lịch trình này có thể được định nghĩa như sau.

  Definition 2 (Non-delay schedule). *A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing.*

Định nghĩa 1 (Lịch trình không chậm trễ). *1 lịch trình khả thi được gọi là không chậm trễ nếu không có máy nào bị giữ ở chế độ nhàn rỗi trong khi 1 hoạt động đang chờ xử lý.*

Requiring a schedule to be non-delay is equivalent to prohibiting unforced idleness. For many models, including those that allow preemptions & have regular objective functions, there are optimal schedules that are non-delay. For many models considered in this part of book, goal: find an optimal schedule that is non-delay. However, there are models where it may be advantageous to have periods of unforced idleness.

– Yêu cầu 1 lịch trình không bị trì hoãn tương đương với việc cấm tình trạng nhàn rỗi không cưỡng bức. Đối với nhiều mô hình, bao gồm cả những mô hình cho phép chiếm quyền ưu tiên & có hàm mục tiêu đều đặn, có những lịch trình tối ưu không bị trì hoãn. Đối với nhiều mô hình được xem xét trong phần này của cuốn sách, mục tiêu là: tìm 1 lịch trình tối ưu không bị trì hoãn. Tuy nhiên, có những mô hình mà việc có các khoảng thời gian nhàn rỗi không cưỡng bức có thể mang lại lợi thế.

A smaller class of schedules, within class of all non-delay schedules, is class of nonpreemtive non-delay schedules. Nonpreemtive non-delay schedules may lead to some interesting & unexpected anomalies.

– 1 lớp lịch trình nhỏ hơn, nằm trong lớp của tất cả các lịch trình không trì hoãn, là lớp lịch trình không ưu tiên không trì hoãn. Lịch trình không ưu tiên không trì hoãn có thể dẫn đến 1 số bất thường thú vị & bất ngờ.

Example 13 (A scheduling anomaly – 1 sự bất thường trong lịch trình). *Consider an instance of* $P2|prec|C_{\max}$ *with* 10 *jobs & processing times:* $\{p_j\}_{j=1}^{10} = 8, 7, 7, 2, 3, 2, 2, 8, 8, 15$. *Jobs are subject to precedence constraints depicted in Fig. 2.2. Makespan of non-delay schedule depicted in* Fig. 2.3. Gantt charts of non-delay schedules: (a) Original schedule. *is 31 & schedule is clearly optimal.*

– Xét 1 trường hợp $P2|prec|C_{\max}$ *với* 10 *công việc & thời gian xử lý:* $\{p_j\}_{j=1}^{10} = 8, 7, 7, 2, 3, 2, 2, 8, 8, 15$. *Các công việc tuân theo các ràng buộc về thứ tự ưu tiên được mô tả trong Hình 2.2. Makespan của lịch trình không trì hoãn được mô tả trong* Hình 2.3. Biểu đồ Gantt của lịch trình không trì hoãn: (a) Lịch trình ban đầu. *là 31 & lịch trình rõ ràng là tối ưu.*

One would expect: if each 1 of 10 processing times is reduced by 1 time unit, makespan would be < 31. *However, requiring schedule to be non-delay results in schedule depicted in* Fig. 2.3.b: processing times 1 unit shorter *with a makespan of 32.*

– Người ta mong đợi: nếu mỗi 1 trong 10 thời gian xử lý được giảm đi 1 đơn vị thời gian, makespan sẽ là < 31. *Tuy nhiên, việc yêu cầu lịch trình không bị trì hoãn sẽ dẫn đến lịch trình được mô tả trong* Hình 2.3.b: thời gian xử lý ngắn hơn 1 đơn vị *với makespan là 32.*

Suppose that an additional machine is made available & that there are now 3 machines instead of 2. One would again expect makespan with original set of processing times to be < 31. *Again, non-delay requirement has an unexpected effect: makespan is now 36.*

– Giả sử có thêm 1 máy & giờ có 3 máy thay vì 2. Ta lại kỳ vọng makespan với tập thời gian xử lý ban đầu là < 31. *1 lần nữa, yêu cầu không trì hoãn lại có tác dụng bất ngờ: makespan giờ là 36.*

Some heuristic procedures & algorithms for job shops are based on construction of nonpreemptive schedules with certain special properties. 2 classes of nonpreemptive schedules are of importance for certain algorithmic procedures for job shops.

– 1 số thủ tục & thuật toán tìm kiếm cho các xưởng gia công dựa trên việc xây dựng các lịch trình không ưu tiên với 1 số thuộc tính đặc biệt. 2 lớp lịch trình không ưu tiên có tầm quan trọng đối với 1 số thủ tục thuật toán cho các xưởng gia công.

Definition 3 (Active schedule). *A feasible nonpreemtive schedule is called* active *if it is not possible to construct another schedule, through changes in order of processing on machines, with at least 1 operation finishing earlier & no operation finishing later.*

Định nghĩa 2 (Lịch trình chủ động). *1 lịch trình không ưu tiên khả thi được gọi là* active *nếu không thể xây dựng 1 lịch trình khác, thông qua những thay đổi về thứ tự xử lý trên máy, với ít nhất 1 thao tác kết thúc sớm hơn & không có thao tác nào kết thúc muộn hơn.*

I.e., a schedule is active if no operation can be put into an empty hole earlier in schedule while preserving feasibility. A nonpreemptive non-delay schedule has to be active but reverse is not necessarily true. Following example describes a schedule that is active but not non-delay.

– Tức là, 1 lịch trình được coi là hoạt động nếu không có thao tác nào có thể được đưa vào 1 lỗ trống sớm hơn trong lịch trình mà vẫn đảm bảo tính khả thi. 1 lịch trình không ưu tiên, không trì hoãn phải hoạt động, nhưng điều ngược lại không nhất thiết đúng. Ví dụ sau mô tả 1 lịch trình hoạt động nhưng không phải là không trì hoãn.

Example 14 (An active schedule). *Consider a job shop with 3 machines & 2 jobs. Job 1 needs 1 time unit on machine 1 & 3 time units on machine 2. Job 2 needs 2 time units on machine 3 & 3 time units on machine 2. Both jobs have to be processed last on machine 2. Consider schedule which processes job 2 on machine 2 before job 1, see* Fig. 2.4: An active schedule that is not non-delay. *Clear: this schedule is active; reversing sequence of 2 jobs on machine 2 postpones processing of job 2. However, schedule is not non-delay. Machine 2 remains idle till time 2, while there is already a job available for processing at time 1.*

– Hãy xem xét 1 xưởng gia công có 3 máy & 2 công việc. Công việc 1 cần 1 đơn vị thời gian trên máy 1 & 3 đơn vị thời gian trên máy 2. Công việc 2 cần 2 đơn vị thời gian trên máy 3 & 3 đơn vị thời gian trên máy 2. Cả hai công việc đều phải được xử lý sau cùng trên máy 2. Hãy xem xét lịch trình xử lý công việc 2 trên máy 2 trước công việc 1, xem Hình 2.4: 1 lịch trình đang hoạt động không phải là không trễ. *Rõ ràng: lịch trình này đang hoạt động; đảo ngược trình tự 2 công việc trên*

máy 2 sẽ hoãn việc xử lý công việc 2. Tuy nhiên, lịch trình này không phải là không trễ. Máy 2 vẫn nhàn rỗi cho đến thời điểm 2, trong khi đã có 1 công việc có sẵn để xử lý tại thời điểm 1.

Can be shown: when objective γ is regular, there exists for $Jm||\gamma$ an optimal schedule that is active.

– Có thể chứng minh: khi mục tiêu γ là chính quy, tồn tại đối với $Jm||\gamma$ 1 lịch trình tối ưu đang hoạt động.

Definition 4 (Semi-active schedule)**.** *A feasible nonpreemptive schedule is called* semi-active *if no operation can be completed earlier without changing order of processing on any 1 of machines.*

Định nghĩa 3 (Lịch trình bán chủ động)**.** *1 lịch trình không ưu tiên khả thi được gọi là* bán chủ động *nếu không có thao tác nào có thể hoàn thành sớm hơn mà không thay đổi thứ tự xử lý trên bất kỳ máy nào.*

Clear: an active schedule has to be semi-active. However, reverse is not necessarily true.

– Rõ ràng: 1 lịch trình hoạt động phải ở trạng thái bán chủ động. Tuy nhiên, điều ngược lại không nhất thiết đúng.

Example 15 (A semi-active schedule)**.** *Consider again a job shop with 3 machines & 2 jobs. Routing of 2 jobs is same as in prev example. Processing times of job 1 on machines 1 & 2 are both equal to 1. Processing times of job 2 on machines 2 & 3 are both equal to 2. Consider schedule under which job 2 is processed on machine 2 before job 1* Fig. 2.5: A semi-active schedule that is not active.*. This implies that job 2 starts its processing on machine 2 at time 2 & job 1 starts its processing on machine 2 at time 4. This schedule is semi-active. However, it is not active, as job 1 can be processed on machine 2 without delaying processing of job 2 on machine 2.*

– Hãy xem xét lại 1 xưởng gia công với 3 máy & 2 công việc. Định tuyến của 2 công việc giống như trong ví dụ trước. Thời gian xử lý của công việc 1 trên máy 1 & 2 đều bằng 1. Thời gian xử lý của công việc 2 trên máy 2 & 3 đều bằng 2. Xem xét lịch trình trong đó công việc 2 được xử lý trên máy 2 trước công việc 1 Hình 2.5: 1 lịch trình bán chủ động không hoạt động.*. Điều này ngụ ý rằng công việc 2 bắt đầu xử lý trên máy 2 tại thời điểm 2 & công việc 1 bắt đầu xử lý trên máy 2 tại thời điểm 4. Lịch trình này là bán chủ động. Tuy nhiên, nó không hoạt động, vì công việc 1 có thể được xử lý trên máy 2 mà không làm chậm quá trình xử lý của công việc 2 trên máy 2.*

An example of a schedule that is not even semi-active can be constructed easily. Postpone start of processing of job 1 on machine 2 for 1 time unit, i.e., machine 2 is kept idle for 1 unit of time between processing of jobs 2 & 1. Clearly, this schedule is not even semi-active.

– 1 ví dụ về 1 lịch trình thậm chí không bán chủ động có thể được xây dựng dễ dàng. Hoãn việc bắt đầu xử lý công việc 1 trên máy 2 trong 1 đơn vị thời gian, tức là, máy 2 được giữ ở chế độ nhàn rỗi trong 1 đơn vị thời gian giữa hai lần xử lý công việc 2 & 1. Rõ ràng, lịch trình này thậm chí không bán chủ động.

Fig. 2.6: Venn diagram of classes of nonpreemptive schedules for job shops. shows a Venn diagram of 3 classes of nonpreemptive schedules: nonpreemptive non-delay schedules, active schedules, & semi-active schedules.

– Hình 2.6: Biểu đồ Venn của các lớp lịch trình không ưu tiên cho các xưởng gia công. cho thấy biểu đồ Venn của 3 lớp lịch trình không ưu tiên: lịch trình không ưu tiên không trì hoãn, lịch trình chủ động & lịch trình bán chủ động.

- 2.4. Complexity Hierarchy. Often, an algorithm for 1 scheduling problem can be applied to another scheduling problem as well. E.g., $1||\sum C_j$ is a special case of $1||\sum w_jC_j$ & a procedure for $1||\sum w_jC_j$ can, of course, also be used for $1||\sum C_j$. In complexity terminology it is then said: $1||\sum C_j$ reduces to $1||\sum w_jC_j$, usually denoted by

$$1||\sum C_j \propto 1||\sum w_jC_j.$$

Based on this concept a chain of reductions can be established. E.g.

$$1||\sum C_j \propto 1||\sum w_jC_j \propto Pm||\sum w_jC_j \propto Qm|prec|\sum w_jC_j.$$

Of course, there are also many problems that are not comparable with one another. E.g., $Pm||\sum w_jT_j$ is not comparable to $Jm||C_{\max}$.

– Phân cấp Độ phức tạp. Thông thường, 1 thuật toán cho 1 bài toán lập lịch cũng có thể được áp dụng cho 1 bài toán lập lịch khác. Ví dụ: $1||\sum C_j$ là 1 trường hợp đặc biệt của $1||\sum w_jC_j$ & 1 thủ tục cho $1||\sum w_jC_j$, tất nhiên, cũng có thể được sử dụng cho $1||\sum C_j$. Trong thuật ngữ độ phức tạp, khi đó ta nói: $1||\sum C_j$ rút gọn thành $1||\sum w_jC_j$, thường được ký hiệu là

$$1||\sum C_j \propto 1||\sum w_jC_j.$$

Dựa trên khái niệm này, 1 chuỗi các phép rút gọn có thể được thiết lập. Ví dụ:

$$1||\sum C_j \propto 1||\sum w_jC_j \propto Pm||\sum w_jC_j \propto Qm|prec|\sum w_jC_j.$$

Tất nhiên, cũng có nhiều bài toán không thể so sánh với nhau. Ví dụ, $Pm||\sum w_jT_j$ không thể so sánh với $Jm||C_{\max}$.

A considerable effort has been made to establish a problem hierarchy describing relationships between hundreds of scheduling problems. In comparisons between complexities of different scheduling problems it is of interest to know how a change in a single element in classification of a problem affects its complexity. In Fig. 2.7: Complexity hierarchies of deterministic scheduling problems: (a) Machine environments. (b) Processing restrictions & constraints. (c) Objective functions. a number of graphs are

exhibited that help determine complexity hierarchy of deterministic scheduling problems. Most of hierarchy depicted in these graphs is relatively straightforward. However, 2 of relationships may need some explaining, namely

$$\alpha|\beta|L_{\max} \propto \alpha|\beta|\sum U_j,\ \alpha|\beta|L_{\max} \propto \alpha|\beta|\sum T_j.$$

It can, indeed, be shown: a procedure for $\alpha|\beta|\sum U_j$ & a procedure for $\alpha|\beta|L_{\max} \propto \alpha|\beta|\sum T_j$ can be applied to $\alpha|\beta|L_{\max}$ with only minor modifications (Exercise 2.23).

– 1 nỗ lực đáng kể đã được thực hiện để thiết lập 1 hệ thống phân cấp vấn đề mô tả mối quan hệ giữa hàng trăm bài toán lập lịch. Khi so sánh độ phức tạp của các bài toán lập lịch khác nhau, điều quan trọng là tìm hiểu xem sự thay đổi trong 1 phần tử duy nhất trong phân loại của 1 bài toán ảnh hưởng đến độ phức tạp của nó như thế nào. Trong Hình 2.7: Hệ thống phân cấp độ phức tạp của các bài toán lập lịch xác định: (a) Môi trường máy. (b) Các hạn chế xử lý & ràng buộc. (c) Hàm mục tiêu., 1 số đồ thị được trình bày giúp xác định hệ thống phân cấp độ phức tạp của các bài toán lập lịch xác định. Hầu hết hệ thống phân cấp được mô tả trong các đồ thị này tương đối đơn giản. Tuy nhiên, 2 trong số các mối quan hệ có thể cần được giải thích, cụ thể là

$$\alpha|\beta|L_{\max} \propto \alpha|\beta|\sum U_j,\ \alpha|\beta|L_{\max} \propto \alpha|\beta|\sum T_j.$$

Thực vậy, có thể chứng minh: 1 thủ tục cho $\alpha|\beta|\sum U_j$ & 1 thủ tục cho $\alpha|\beta|L_{\max} \propto \alpha|\beta|\sum T_j$ có thể được áp dụng cho $\alpha|\beta|L_{\max}$ chỉ với 1 số sửa đổi nhỏ (Bài tập 2.23).

A significant amount of research in deterministic scheduling has been devoted to finding efficient, so-called polynomial time, algorithms for scheduling problems. However, many scheduling problems do not have a polynomial time algorithm; these problems are so-called *NP-hard* problems. Verifying that a problem is NP-hard requires a formal mathematical proof (see Appendix D).

– 1 lượng lớn nghiên cứu về lập lịch xác định đã được dành cho việc tìm kiếm các thuật toán hiệu quả, được gọi là thuật toán thời gian đa thức, cho các bài toán lập lịch. Tuy nhiên, nhiều bài toán lập lịch không có thuật toán thời gian đa thức; những bài toán này được gọi là bài toán *NP-hard*. Việc xác minh 1 bài toán là NP-hard đòi hỏi 1 chứng minh toán học chính thức (xem Phụ lục D).

Research in past has focused in particular on borderline between polynomial time solvable problems & NP-hard problems. E.g., in string of problems described above, $1||\sum w_jC_j$ can be solved in polynomial time, whereas $Pm||\sum w_jC_j$ is NP-hard, which implies that $Qm|prec|\sum w_jC_j$ is also NP-hard. Following examples illustrate borderlines between easy & hard problems within given sets of problems.

– Nghiên cứu trước đây đặc biệt tập trung vào ranh giới giữa các bài toán có thể giải được trong thời gian đa thức & bài toán NP-khó. Ví dụ, trong chuỗi bài toán được mô tả ở trên, $1||\sum w_jC_j$ có thể được giải trong thời gian đa thức, trong khi $Pm||\sum w_jC_j$ là NP-khó, ngụ ý rằng $Qm|prec|\sum w_jC_j$ cũng là NP-khó. Các ví dụ sau minh họa ranh giới giữa các bài toán dễ & khó trong các tập bài toán cho trước.

Example 16 (A complexity hierarchy)**.** *Consider problems:* $1||C_{\max}, P2||C_{\max}, F2||C_{\max}, Jm||C_{\max}, FFc||C_{\max}$. Fig. 2.8: Complexity hierarchy of problems in Example 2.4.1.

Example 17 (A complexity hierarchy)**.** *Consider problems:* $1||L_{\max}, 1|prmp|L_{\max}, 1|r_j|L_{\max}, 1|r_j, prmp|L_{\max}, Pm||L_{\max}$. Fig. 2.8: Complexity hierarchy of problems in Example 2.4.2.

- 3. Single Machine Models (Deterministic).
- 4. Advanced Single Machine Models (Deterministic).
- 5. Parallel Machine Models (Deterministic).
- 6. Flow Shops & Flexible Flow Shops (Deterministic).
- 7. Job Shops (Deterministic). This chap deals with multi-operation models that are different from flow shop models discussed in previous chap. In a flow shop model all jobs follow the same route. When the routes are fixed, but not necessarily the same for each job, the model is called a job shop. If a job in a job shop has to visit certain machines more than once, the job is said to recirculate. Recirculation is a common phenomenon in the real world. e.g., in semiconductor manufacturing jobs have to recirculate several times before they complete all their processing.

  – Chương này đề cập đến các mô hình đa tác vụ khác với các mô hình xưởng sản xuất đã được thảo luận trong chương trước. Trong mô hình xưởng sản xuất, tất cả các công việc đều theo cùng 1 lộ trình. Khi các lộ trình được cố định, nhưng không nhất thiết phải giống nhau cho mỗi công việc, mô hình này được gọi là xưởng gia công. Nếu 1 công việc trong xưởng gia công phải đi qua 1 số máy móc nhất định nhiều hơn 1 lần, công việc đó được gọi là tuần hoàn. Tuần hoàn là 1 hiện tượng phổ biến trong thế giới thực. Ví dụ, trong sản xuất chất bán dẫn, các công việc phải tuần hoàn nhiều lần trước khi hoàn tất toàn bộ quá trình xử lý.

  1st sect focuses on representations & formulations of classical job shop problem with makespan objective & no recirculation. It also describes a branch-&-bound procedure that is designed to find the optimal solution. 2nd sect describes a popular heuristic for job shops with makespan objective & no recirculation. This heuristic is typically referred to as *Shifting Bottleneck* heuristic. 3rd sect focuses on a more elaborate version of shifting bottleneck heuristic that is designed specifically for total weighted tardiness objective. 4th sect describes an application of a constraint programming procedure for minimization of makespan. Last sect discusses possible extensions.

– Phần 1 tập trung vào các biểu diễn & công thức của bài toán xưởng công việc cổ điển với mục tiêu makespan & không có sự tuần hoàn. Phần này cũng mô tả 1 thủ tục ràng buộc nhánh được thiết kế để tìm ra giải pháp tối ưu. Phần 2 mô tả 1 phương pháp heuristic phổ biến cho các xưởng công việc với mục tiêu makespan & không có sự tuần hoàn. Phương pháp heuristic này thường được gọi là phương pháp heuristic *Shifting Bottleneck*. Phần 3 tập trung vào 1 phiên bản phức tạp hơn của phương pháp heuristic nút cổ chai dịch chuyển được thiết kế riêng cho mục tiêu trễ có trọng số tổng thể. Phần 4 mô tả 1 ứng dụng của thủ tục lập trình ràng buộc để tối thiểu hóa makespan. Phần cuối cùng thảo luận về các khả năng mở rộng.

◦ 7.1. Disjunctive Programming & Branch & Bound. Consider $J2||C_{\max}$. There are 2 machines & n jobs. Some jobs have to be processed 1st on machine 1 & then on machine 2, while the remaining jobs have to be processed 1st on machine 2 & then on machine 1. Processing time of job j on machine 1 & 2 are p_{1j}, p_{2j}. Objective: minimize makespan.

– Lập trình Phân biệt & Nhánh & Giới hạn. Xét $J2||C_{\max}$. Có 2 máy & n công việc. 1 số công việc phải được xử lý trước trên máy 1 & sau đó trên máy 2, trong khi các công việc còn lại phải được xử lý trước trên máy 2 & sau đó trên máy 1. Thời gian xử lý của công việc j trên máy 1 & 2 là p_{1j}, p_{2j}. Mục tiêu: giảm thiểu makespan.

This problem can be reduced to $F2||C_{\max}$ as follows. Let $J_{1,2}$ denote set of jobs that have to be processed 1st on machine 1, & $J_{2.1}$ set of jobs that have to be processed 1st on machine 2. Observe that when a job from $J_{1,2}$ has completed its processing on machine 1, postponing its processing on machine 2 does not affect affect makespan as long as machine 2 is kept busy. Same can be said about a job from $J_{2,1}$; if such a job has completed its processing on machine 2, postponing its processing on machine 1 (as long as machine 1 is kept busy) does not affect makespan. Hence, a job from $J_{1,2}$ has on machine 1 a higher priority than any job from $J_{2,1}$, while a job from $J_{2,1}$ has on machine 2 a higher priority than any job from $J_{1,2}$. It remains to be determined in what sequence jobs in $J_{1,2}$ go through machine 1 & jobs in $J_{2,1}$ go through machine 2. 1st of these 2 sequences can be determined by considering $J_{1,2}$ as an $F2||C_{\max}$ problem with machine 1 set up 1st & machine 2 set up 2nd, & 2nd sequence can be determined by considering $J_{2,1}$ as another $F2||C_{\max}$ problem with machine 2 set up 1st & machine 1 2nd. This leads to SPT(1)-LPT(2) sequences for each of 2 sets, with priorities between sets as specified above.

– Bài toán này có thể được rút gọn thành $F2||C_{\max}$ như sau. Giả sử $J_{1,2}$ biểu thị tập hợp các công việc phải được xử lý đầu tiên trên máy 1, & $J_{2.1}$ tập hợp các công việc phải được xử lý đầu tiên trên máy 2. Lưu ý rằng khi 1 công việc từ $J_{1,2}$ đã hoàn tất việc xử lý của nó trên máy 1, thì việc hoãn xử lý của nó trên máy 2 không ảnh hưởng đến makespan miễn là máy 2 vẫn bận. Tương tự cũng có thể nói về 1 công việc từ $J_{2,1}$; nếu 1 công việc như vậy đã hoàn tất việc xử lý của nó trên máy 2, thì việc hoãn xử lý của nó trên máy 1 (miễn là máy 1 vẫn bận) không ảnh hưởng đến makespan. Do đó, 1 công việc từ $J_{1,2}$ có trên máy 1 mức độ ưu tiên cao hơn bất kỳ công việc nào từ $J_{2,1}$, trong khi 1 công việc từ $J_{2,1}$ có trên máy 2 mức độ ưu tiên cao hơn bất kỳ công việc nào từ $J_{1,2}$. Vẫn chưa xác định được trình tự nào mà các công việc trong $J_{1,2}$ đi qua máy 1 & các công việc trong $J_{2,1}$ đi qua máy 2. Trình tự đầu tiên trong 2 trình tự này có thể được xác định bằng cách coi $J_{1,2}$ là 1 vấn đề $F2||C_{\max}$ với máy 1 thiết lập thứ nhất & máy 2 thiết lập thứ hai, trình tự & thứ hai có thể được xác định bằng cách coi $J_{2,1}$ là 1 vấn đề $F2||C_{\max}$ khác với máy 2 thiết lập thứ nhất & máy 1 thứ hai. Điều này dẫn đến chuỗi SPT(1)-LPT(2) cho mỗi tập hợp trong 2 tập hợp, với các ưu tiên giữa các tập hợp như đã chỉ định ở trên.

This 2-machine problem is 1 of few job shop scheduling problems for which a polynomial time algorithm can be found. Few other JSSPs for which polynomial time algorithms can be obtained usually require all processing times to be either 0 or 1. Remainder of this sect is dedicated to $Jm||C_{\max}$ problem with arbitrary processing times & no recirculation.

– Bài toán 2 máy này là 1 trong số ít bài toán lập lịch xưởng gia công mà thuật toán thời gian đa thức có thể được tìm thấy. 1 số JSSP khác mà thuật toán thời gian đa thức có thể tìm thấy thường yêu cầu tất cả thời gian xử lý bằng 0 hoặc 1. Phần còn lại của phần này dành riêng cho bài toán $Jm||C_{\max}$ với thời gian xử lý tùy ý & không có tuần hoàn.

Minimizing makespan in a job shop without recirculation, $Jm||C_{\max}$, can be represented in a very nice way by a disjunctive graph. Consider a directed graph G with a set of nodes N & 2 sets of arcs A, B. Nodes N correspond to all operations (i, j) that must be performed on n jobs. So-called *conjunctive* (solid) arcs A represent routes of jobs. If arc $(i, j) \to (k, j)$ is part of A, then job j has to be processed on machine i before it is processed on machine k, i.e., operation (i, j) precedes operation (k, j). 2 operations that belong to 2 different jobs & that have to be processed on same machine are connected to one another by 2 so-called *disjunctive* (broken) arcs that go in opposite directions. Disjunctive arcs B form m cliques of double arcs, 1 clique for each machine. (A clique is a term in graph theory that refers to a graph in which any 2 nodes are connected to 1 another; in this case each connection within a clique consists of a pair of disjunctive arcs.) All operations (nodes) in same clique have to be done on same machine. All arcs emanating from a node, conjunctive as well as disjunctive, have as length processing time of operation that is represented by that node. In addition, there is a source U & a sink V, which are dummy nodes. Source node U has n conjunctive arcs emanating to 1st operations of n jobs & sink node V has n conjunctive arcs coming in from all last operations. Arcs emanating from source have length 0 Fig. 7.1: Directed job for job shop with makespan as objective. This graph is denoted by $G = (N, A, B)$.

– Việc tối thiểu hóa makespan trong 1 xưởng gia công mà không cần tuần hoàn, $Jm||C_{\max}$, có thể được biểu diễn 1 cách rất hay bằng 1 đồ thị rời rạc. Xét 1 đồ thị có hướng G với 1 tập các nút N & 2 tập các cung A, B. Các nút N tương ứng với tất cả các thao tác (i, j) phải được thực hiện trên n công việc. Các cung *conjunctive* (liên tiếp) A biểu diễn các tuyến công việc. Nếu cung $(i, j) \to (k, j)$ là 1 phần của A, thì công việc j phải được xử lý trên máy i trước khi được xử lý trên máy k, tức là thao tác (i, j) diễn ra trước thao tác (k, j). 2 thao tác thuộc 2 công việc khác nhau & phải được xử lý trên cùng 1 máy được kết nối với nhau bằng 2 cung được gọi là *disjunctive* (bị gãy) đi theo hướng ngược nhau. Các cung disjunctive B tạo thành m clique gồm các cung đôi, 1 clique cho mỗi máy. (1 clique là 1 thuật ngữ trong lý thuyết đồ thị dùng để chỉ 1 đồ thị trong đó bất kỳ 2 nút nào được kết nối với 1 nút khác; trong trường hợp này, mỗi kết nối trong 1 clique bao gồm 1 cặp cung disjunctive.) Tất cả các thao tác (nút) trong cùng 1 clique phải được thực hiện trên cùng 1 máy. Tất cả các cung phát ra từ 1 nút, cả hợp & rời, đều có thời gian xử lý thao tác được biểu diễn bởi nút đó. Ngoài ra, còn có 1 nguồn U & 1

đích V, là các nút giả. Nút nguồn U có n cung liên kết xuất phát từ thao tác đầu tiên của n công việc & nút đích V có n cung liên kết xuất phát từ tất cả các thao tác cuối cùng. Các cung xuất phát từ nguồn có độ dài 0 Hình 7.1: Công việc được định hướng cho xưởng gia công với makespan là mục tiêu. Đồ thị này được ký hiệu là $G = (N, A, B)$.

A feasible schedule corresponds to a *selection* of 1 disjunctive arc from each pair s.t. resulting directed graph is acyclic. This implies that a selection of disjunctive arcs from a clique has to be acyclic. Such a selection determines sequence in which operations are to be performed on that machine. That a selection from a clique has to be acyclic can be argued as follows: If there were a cycle within a clique, a feasible sequence of operations on corresponding machine would not have been possible. It may not be immediately obvious why there should not be any cycle formed by conjunctive arcs & disjunctive arcs from different cliques. However, such a cycle would correspond also to a situation that is infeasible. E.g., let $(h, j), (i, j)$ denote 2 consecutive operations that belong to job j, & let $(i, k), (h, k)$ denote 2 consecutive operations that belong to job k. If under a given schedule operation (i, j) precedes operation (i, k) on machine i & operation (h, k) precedes operation (h, j) on machine h, then graph contains a cycle with 4 arcs, 2 conjunctive arcs & 2 disjunctive arcs from different cliques. Such a schedule is physically impossible. Summarizing, if D denotes subset of selected disjunctive arcs & graph $G(D)$ is defined by set of conjunctive arcs & subset D, then D corresponds to a feasible schedule iff $G(D)$ contains no directed cycles.

– 1 lịch trình khả thi tương ứng với 1 *selection* của 1 cung rời rạc từ mỗi cặp s.t. đồ thị có hướng kết quả là phi chu trình. Điều này ngụ ý rằng 1 lựa chọn các cung rời rạc từ 1 clique phải là phi chu trình. 1 lựa chọn như vậy xác định trình tự các hoạt động sẽ được thực hiện trên máy đó. Việc lựa chọn từ 1 clique phải là phi chu trình có thể được lập luận như sau: Nếu có 1 chu trình trong 1 clique, thì 1 trình tự hoạt động khả thi trên máy tương ứng sẽ không thể thực hiện được. Có thể không rõ ràng ngay lập tức tại sao lại không có bất kỳ chu trình nào được hình thành bởi các cung nối & các cung rời rạc từ các clique khác nhau. Tuy nhiên, 1 chu trình như vậy cũng sẽ tương ứng với 1 tình huống không khả thi. Ví dụ: giả sử $(h, j), (i, j)$ biểu thị 2 hoạt động liên tiếp thuộc về công việc j, & giả sử $(i, k), (h, k)$ biểu thị 2 hoạt động liên tiếp thuộc về công việc k. Nếu theo 1 lịch trình cho trước, thao tác (i, j) diễn ra trước thao tác (i, k) trên máy i & thao tác (h, k) diễn ra trước thao tác (h, j) trên máy h, thì đồ thị chứa 1 chu trình với 4 cung, 2 cung liên hợp & 2 cung rời rạc từ các nhóm khác nhau. 1 lịch trình như vậy là không thể về mặt vật lý. Tóm lại, nếu D biểu thị tập con của các cung rời rạc được chọn & đồ thị $G(D)$ được định nghĩa bởi tập các cung liên hợp & tập con D, thì D tương ứng với 1 lịch trình khả thi nếu & chỉ nếu $G(D)$ không chứa chu trình có hướng nào.

Makespan of a feasible schedule is determined by longest path in $G(D)$ from source U to sink V. This longest path consists of a set of operations of which the 1st starts at time 0 & the last finishes at time of makespan. Each operation on this path is immediately followed by either next operation on same machine or next operation of same job on another machine. Problem of minimizing makespan is reduced to finding a selection of disjunctive arcs that minimizes length of longest path (i.e., *critical* path).

– Makespan của 1 lịch trình khả thi được xác định bởi đường đi dài nhất trong $G(D)$ từ nguồn U đến đích V. Đường đi dài nhất này bao gồm 1 tập hợp các thao tác, trong đó thao tác đầu tiên bắt đầu tại thời điểm 0 & thao tác cuối cùng kết thúc tại thời điểm makespan. Mỗi thao tác trên đường đi này được theo sau ngay lập tức bởi thao tác tiếp theo trên cùng máy hoặc thao tác tiếp theo của cùng 1 công việc trên 1 máy khác. Bài toán tối thiểu hóa makespan được rút gọn thành việc tìm 1 tập hợp các cung rời rạc sao cho tối thiểu hóa độ dài của đường đi dài nhất (tức là đường tới hạn).

There are several mathematical programming formulations for job shop without recirculation, including a number of integer programming formulations. However, formulation most often used is so-called disjunctive programming formulation (see also Appendix A). This disjunctive programming formulation is closely related to disjunctive graph representation of job shop.

– Có 1 số công thức lập trình toán học cho job shop không có tuần hoàn, bao gồm 1 số công thức lập trình số nguyên. Tuy nhiên, công thức được sử dụng phổ biến nhất là công thức lập trình phân biệt (xem thêm Phụ lục A). Công thức lập trình phân biệt này có liên quan chặt chẽ đến biểu diễn đồ thị phân biệt của job shop.

To present disjunctive programming formulation, let variable y_{ij} denote starting time of operation (i, j). Recall that set N denotes set of all operations (i, j), & set A set of all routing constraints $(i, j) \to (k, j)$ that require job j to be processed on machine i before it is processed on machine k. Following mathematical program minimizes makespan:

– Để trình bày công thức lập trình phân biệt, hãy cho biến y_{ij} biểu thị thời điểm bắt đầu của thao tác (i, j). Nhớ lại rằng tập N biểu thị tập hợp tất cả các thao tác (i, j), & tập A biểu thị tập hợp tất cả các ràng buộc định tuyến $(i, j) \to (k, j)$ yêu cầu công việc j phải được xử lý trên máy i trước khi được xử lý trên máy k. Chương trình toán học sau đây tối thiểu hóa makespan:

$$\text{minimize } C_{\max} \text{ subject to } \begin{cases} y_{kj} - y_{ij} \geq p_{ij} & \forall (i,j) \to (k,j) \in A, \\ C_{\max} - y_{ij} \geq p_{ij} & \forall (i,j) \in N, \\ y_{ij} - y_{il} \geq p_{il} \text{ or } y_{il} - y_{ij} \geq p_{ij} & \forall (i,l),(i,j), i \in [m], \\ y_{ij} \geq 0 & \forall (i,j) \in N. \end{cases}$$

In this formulation, 1st set of constraints ensure that operation (k, j) cannot start before operation (i, j) is completed. 3rd set of constraints are called *disjunctive constraints*; they ensure that some ordering exists among operations of different jobs that have to be processed on same machine. Because of these constraints, this formulation is referred to as a disjunctive programming formulation.

– Trong công thức này, tập ràng buộc thứ nhất đảm bảo rằng thao tác (k, j) không thể bắt đầu trước khi thao tác (i, j) hoàn tất. Tập ràng buộc thứ ba được gọi là *ràng buộc rời rạc*; chúng đảm bảo rằng có 1 số thứ tự tồn tại giữa các thao tác

của các công việc khác nhau phải được xử lý trên cùng 1 máy. Do những ràng buộc này, công thức này được gọi là công thức lập trình rời rạc.

Example 18 (Disjunctive Programming Formulation – Công thức lập trình/quy hoạch rời rạc)**.** *Consider following example with 4 machines & 3 jobs. Route, i.e., machine sequence, as well as processing times are given in table* Table: jobs | machine sequence | processing times. *Objective consists of single variable* $C_{\max}$. *1st set of constraints consists of 7 constraints: 2 for job 1, 3 for job 2, & 2 for job 3. E.g., 1 of these is* $y_{21} - y_{11} \geq p_{11} = 10$. *2nd set consists of 10 constraints, 1 for each operation. An example is* $C_{\max} - y_{11} \geq p_{11} = 10$. *Set of disjunctive constraints contains 8 constraints: 3 each for machines 1 & 2 & 1 each for machines 3 & 4 (there are 3 operations to be performed on machines 1 & 2 & 2 operations on machines 3 & 4). An example of a disjunctive constraint is* $y_{11} - y_{12} \geq p_{12} = 3$ *or* $y_{12} - y_{11} \geq p_{11} = 10$. *Last set includes 10 nonnegativity constraints, 1 for each starting time.*

– *Xét ví dụ sau với 4 máy & 3 công việc. Tuyến đường, tức là trình tự máy, cũng như thời gian xử lý được đưa ra trong bảng* Bảng: công việc | trình tự máy | thời gian xử lý. *Mục tiêu bao gồm 1 biến duy nhất* $C_{\max}$. *Bộ ràng buộc thứ nhất bao gồm 7 ràng buộc: 2 cho công việc 1, 3 cho công việc 2, & 2 cho công việc 3. Ví dụ: 1 trong số này là* $y_{21} - y_{11} \geq p_{11} = 10$. *Bộ ràng buộc thứ hai bao gồm 10 ràng buộc, 1 cho mỗi thao tác. Ví dụ:* $C_{\max} - y_{11} \geq p_{11} = 10$. *Tập hợp các ràng buộc phân biệt chứa 8 ràng buộc: 3 ràng buộc cho mỗi máy 1 & 2 & 1 ràng buộc cho mỗi máy 3 & 4 (có 3 thao tác cần thực hiện trên máy 1 & 2 & 2 thao tác trên máy 3 & 4). 1 ví dụ về ràng buộc phân biệt là* $y_{11} - y_{12} \geq p_{12} = 3$ *hoặc* $y_{12} - y_{11} \geq p_{11} = 10$. *Tập hợp cuối cùng bao gồm 10 ràng buộc không âm, 1 ràng buộc cho mỗi thời điểm bắt đầu.*

A scheduling problem can be formulated as a disjunctive program does not imply that there is a standard solution procedure available that will work satisfactorily. Minimizing makespan in a job shop is a very hard problem & solution procedures are either based on enumeration or heuristics.

– 1 bài toán lập lịch có thể được xây dựng dưới dạng 1 chương trình rời rạc không ngụ ý rằng có 1 quy trình giải chuẩn có thể hoạt động hiệu quả. Việc tối thiểu hóa thời gian chờ (makespan) trong 1 xưởng gia công là 1 bài toán rất khó & các quy trình giải thường dựa trên phép liệt kê hoặc phương pháp tìm kiếm.

To obtain optimal solutions, branch-&-bound methods are required. Branching as well as bounding procedures that are applicable to this problem are usually of a special design. In order to describe 1 of branching procedures, a specific class of schedules is considered.

– Để có được các giải pháp tối ưu, cần sử dụng các phương pháp nhánh-&-bound. Các thủ tục phân nhánh cũng như giới hạn áp dụng cho bài toán này thường được thiết kế đặc biệt. Để mô tả 1 trong các thủ tục phân nhánh, cần xem xét 1 lớp lịch trình cụ thể.

Definition 5 (Active schedule)**.** *A feasible schedule is called* active *if it cannot be altered in any way such that some operation is completed earlier & no other operation is completed later.*

Định nghĩa 4 (Lịch trình chủ động)**.** *1 lịch trình khả thi được gọi là chủ động nếu không thể thay đổi theo bất kỳ cách nào để 1 số hoạt động được hoàn thành sớm hơn & không có hoạt động nào khác được hoàn thành muộn hơn.*

A schedule being active implies that when a job arrives at a machine, this job is processed in prescribed sequence as early as possible. An active schedule cannot have any idle period in which operation of a waiting job could fit.

– 1 lịch trình đang hoạt động ngụ ý rằng khi 1 công việc đến máy, công việc đó sẽ được xử lý theo trình tự quy định càng sớm càng tốt. 1 lịch trình đang hoạt động không thể có bất kỳ khoảng thời gian nhàn rỗi nào phù hợp với hoạt động của 1 công việc đang chờ.

From definition, it follows that an active schedule has property that impossible to reduce makespan without increasing starting time of some operation. Of course, there are many different active schedules. It can be shown that there exists among all possible schedules an active schedule that minimizes makespan.

– Từ định nghĩa, ta thấy rằng 1 lịch trình chủ động có đặc tính là không thể giảm makespan mà không làm tăng thời gian bắt đầu của 1 thao tác nào đó. Tất nhiên, có rất nhiều lịch trình chủ động khác nhau. Có thể chứng minh rằng trong tất cả các lịch trình khả dĩ, luôn tồn tại 1 lịch trình chủ động tối thiểu hóa makespan.

A branching scheme often used is based on generation of all active schedules. All such active schedules can be generated by a simple algorithm. In this algorithm, Ω denotes set of all operations of which all predecessors already have been scheduled (i.e., set of all schedulable operations) & r_{ij} earliest possible starting time of operation (i, j) in Ω. Set $\Omega' \subset \Omega$.

– 1 sơ đồ phân nhánh thường được sử dụng dựa trên việc tạo ra tất cả các lịch trình đang hoạt động. Tất cả các lịch trình đang hoạt động như vậy có thể được tạo ra bằng 1 thuật toán đơn giản. Trong thuật toán này, Ω biểu thị tập hợp tất cả các thao tác mà tất cả các thao tác tiền nhiệm đã được lên lịch (tức là tập hợp tất cả các thao tác có thể lên lịch) & r_{ij} thời điểm bắt đầu sớm nhất có thể của thao tác (i, j) trong Ω. $\Omega' \subset \Omega$.

Algorithm 7.1.3 (Generation of all Active Schedules).

1. Initial Condition: Let Ω contains 1st operation of each job, let $r_{ij} = 0$, $\forall (i, j) \in \Omega$.
2. Machine Selection: Compute for current partial schedule

$$t(\Omega) = \min_{(i,j) \in \Omega} r_{ij} + p_{ij}$$

& let i^* denote machine on which minimum is achieved.

3. Branching: Let Ω' denote set of all operations (i^*, j) on machine i^* s.t.

$$r_{i^*j} < t(\Omega).$$

For each operation in Ω' consider an (extended) partial schedule with that operation as next one on machine i^*. For each such (extended) partial schedule delete operation from Ω, include its immediate follower in Ω & return to Step 2.

– **Thuật toán 7.1.3 (Tạo tất cả các Lịch trình Hoạt động).**

1. Điều kiện Ban đầu: Cho Ω chứa thao tác đầu tiên của mỗi công việc, cho $r_{ij} = 0$, $\forall (i, j) \in \Omega$.
2. Lựa chọn Máy: Tính toán cho lịch trình 1 phần hiện tại

$$t(\Omega) = \min_{(i,j)\in\Omega} r_{ij} + p_{ij}$$

& cho i^* biểu thị máy đạt được giá trị tối thiểu.
3. Phân nhánh: Cho Ω' biểu thị tập hợp tất cả các thao tác (i^*, j) trên máy i^* s.t.

$$r_{i^*j} < t(\Omega).$$

Đối với mỗi thao tác trong Ω', hãy xem xét 1 lịch trình 1 phần (mở rộng) với thao tác đó là thao tác tiếp theo trên máy i^*. Đối với mỗi thao tác xóa lịch trình 1 phần (mở rộng) như vậy khỏi Ω, hãy đưa thao tác tiếp theo ngay sau nó vào Ω & quay lại Bước 2.

Algorithm 7.1.3 is basis for branching process. Step 3 performs branching from node that is characterized by given partial schedule; number of branches is equal to number of operations in Ω'. With this algorithm one can generate entire tree & nodes at very bottom of tree correspond to all active schedules.

– Thuật toán 7.1.3 là cơ sở cho quá trình phân nhánh. Bước 3 thực hiện phân nhánh từ nút được đặc trưng bởi 1 lịch trình cục bộ cho trước; số nhánh bằng số phép toán trong Ω'. Với thuật toán này, ta có thể tạo ra toàn bộ cây & các nút ở dưới cùng của cây tương ứng với tất cả các lịch trình đang hoạt động.

So a node $\mathcal{V}$ in tree corresponds to a partial schedule & partial schedule is characterized by a selection of disjunctive arcs that corresponds to order in which all predecessors of a given set Ω have been scheduled. A branch out of node $\mathcal{V}$ corresponds to selection of an operation $(i^*, j) \in \Omega'$ as next one to go on machine i^*. This implies that newly created node at lower level, say node $\mathcal{V}'$, which corresponds to a partial schedule with only 1 more operation in place, contains various additional disjunctive arcs that are now selected (see Fig. 7.2: Branching tree for branch-&-bound approach). Let D' denote set of disjunctive arcs selected at newly created node. Refer to graph that includes all conjunctive arcs & set D' as graph $G(D')$. Number of branches sprouting from node $\mathcal{V}$ is equal to number of operations in Ω'.

– Vì vậy, 1 nút $\mathcal{V}$ trong cây tương ứng với 1 lịch trình 1 phần & lịch trình 1 phần được đặc trưng bởi 1 lựa chọn các cung rời rạc tương ứng với thứ tự mà tất cả các phần tử tiền nhiệm của 1 tập hợp Ω đã cho đã được lên lịch. 1 nhánh ra khỏi nút $\mathcal{V}$ tương ứng với việc lựa chọn 1 thao tác $(i^*, j) \in \Omega'$ là thao tác tiếp theo sẽ thực hiện trên máy i^*. Điều này ngụ ý rằng nút mới được tạo ở cấp thấp hơn, chẳng hạn như nút $\mathcal{V}'$, tương ứng với 1 lịch trình 1 phần chỉ có thêm 1 thao tác tại chỗ, chứa nhiều cung rời rạc bổ sung khác nhau hiện được chọn (xem Hình 7.2: Cây phân nhánh cho phương pháp tiếp cận nhánh-&-bound). Giả sử D' biểu thị tập hợp các cung rời rạc được chọn tại nút mới được tạo. Tham khảo đồ thị bao gồm tất cả các cung nối & đặt D' là đồ thị $G(D')$. Số nhánh mọc ra từ nút $\mathcal{V}$ bằng số phép toán trong Ω'.

To find a lower bound for makespan at node $\mathcal{V}'$, consider graph $G(D')$. Length of critical path in this graph already results in a lower bound for makespan at node $\mathcal{V}'$. Call this lower bound $\mathrm{LB}(\mathcal{V}')$. Better (higher) lower bounds for this node can be obtained as follows.

– Để tìm giới hạn dưới cho makespan tại nút $\mathcal{V}'$, hãy xem xét đồ thị $G(D')$. Độ dài đường dẫn tới hạn trong đồ thị này đã tạo ra giới hạn dưới cho makespan tại nút $\mathcal{V}'$. Gọi giới hạn dưới này là $\mathrm{LB}(\mathcal{V}')$. Có thể tìm được giới hạn dưới tốt hơn (cao hơn) cho nút này như sau.

Consider machine i & assume that all other machines are allowed to process, at any point in time, multiple operations simultaneously (since not all disjunctive arcs have been selected yet in $G(D')$, it may be case: at some points in time, multiple operations require processing on same machine at same time). However, machine i must process its operations 1 after another. 1st, compute earliest possible starting times r_{ij} of all operations (i, j) on machine i; i.e., determine in graph $G(D')$ length of longest path from source to node (i, j). 2nd, for each operation (i, j) on machine i, compute minimum amount of time needed between completion of operation (i, j) & lower bound $\mathrm{LB}(\mathcal{V}')$, by determining longest path from node (i, j) to sink in $G(D')$. This amount of time, together with lower bound on makespan, translates into a due date d_{ij} for operation (i, j), i.e., d_{ij} is equal to $\mathrm{LB}(\mathcal{V}')$ minus length of longest path from node (i, j) to sink plus p_{ij}.

– Xét máy i & giả sử rằng tất cả các máy khác đều được phép xử lý, tại bất kỳ thời điểm nào, nhiều thao tác đồng thời (vì không phải tất cả các cung rời rạc đều đã được chọn trong $G(D')$, nên có thể xảy ra trường hợp: tại 1 số thời điểm, nhiều thao tác yêu cầu xử lý trên cùng 1 máy tại cùng 1 thời điểm). Tuy nhiên, máy i phải xử lý các thao tác của nó lần lượt. 1. Trước tiên, hãy tính thời điểm bắt đầu sớm nhất có thể r_{ij} của tất cả các thao tác (i, j) trên máy i; tức là, xác định trong đồ thị $G(D')$ độ dài đường đi dài nhất từ nguồn đến nút (i, j). 2. Đối với mỗi thao tác (i, j) trên máy i, hãy tính lượng thời gian tối thiểu cần thiết giữa lúc hoàn thành thao tác (i, j) & giới hạn dưới $\mathrm{LB}(\mathcal{V}')$, bằng cách xác định đường đi dài nhất từ nút (i, j) đến điểm dừng trong $G(D')$. Khoảng thời gian này, cùng với giới hạn dưới của makespan, được chuyển thành ngày đến hạn d_{ij} cho hoạt động (i, j), tức là d_{ij} bằng $\mathrm{LB}(\mathcal{V}')$ trừ đi độ dài đường đi dài nhất từ nút (i, j) đến bồn rửa cộng với p_{ij}.

Consider now problem of sequencing operations on machine i as a single machine problem with jobs arriving at different release dates, no preemptions allowed, & maximum lateness as objective to be minimized, i.e., $1|r_j|L_{\max}$ (Sect. 3.2). Even though this problem is strongly NP-hard, there are relatively effective algorithms that generate good solutions. Optimal sequence obtained for this problem implies a selection of disjunctive arcs that can be added (temporarily) to D'. This then may lead to a longer overall critical path in graph, a larger makespan, & a better (higher) lower bound for node $\mathcal{V}'$. At node $\mathcal{V}'$, this can be done for each of m machines separately. Largest makespan obtained this way can be used as a lower bound at node $\mathcal{V}'$. Of course, temporary disjunctive arcs inserted to obtain lower bound are deleted as soon as best lower bound is determined.

– Bây giờ hãy xem xét bài toán sắp xếp các thao tác trên máy i như 1 bài toán máy đơn với các công việc đến vào các ngày phát hành khác nhau, không được phép chiếm dụng trước, & độ trễ tối đa là mục tiêu cần giảm thiểu, tức là $1|r_j|L_{\max}$ (Mục 3.2). Mặc dù bài toán này rất khó, nhưng có những thuật toán tương đối hiệu quả tạo ra các giải pháp tốt. Trình tự tối ưu thu được cho bài toán này ngụ ý 1 lựa chọn các cung rời rạc có thể được thêm (tạm thời) vào D'. Điều này sau đó có thể dẫn đến 1 đường dẫn tới hạn tổng thể dài hơn trong đồ thị, 1 makespan lớn hơn, & 1 giới hạn dưới tốt hơn (cao hơn) cho nút $\mathcal{V}'$. Tại nút $\mathcal{V}'$, điều này có thể được thực hiện riêng cho mỗi máy trong số m máy. Makespan lớn nhất thu được theo cách này có thể được sử dụng làm giới hạn dưới tại nút $\mathcal{V}'$. Tất nhiên, các cung rời rạc tạm thời được chèn vào để có được giới hạn dưới sẽ bị xóa ngay khi giới hạn dưới tốt nhất được xác định.

Although it appears somewhat of a burden to have to solve m strongly NP-hard scheduling problems in order to obtain 1 lower bound for another strongly NP-hard problem, this type of bounding procedure has performed reasonably well in computational experiments.

– Mặc dù có vẻ hơi nặng nề khi phải giải m các bài toán lập lịch NP-khó mạnh để có được 1 giới hạn dưới cho 1 bài toán NP-khó mạnh khác, nhưng loại thủ tục giới hạn này đã hoạt động khá tốt trong các thí nghiệm tính toán.

Example 19 (Application of branch & bound)**.** *Consider instance described in Example 7.1.1. Initial graph contains only conjunctive arcs & is depicted in* Fig. 7.3a: Precedence graphs at Level 1 in Example 7.1.4. *Makespan corresponding to this graph is 22. Applying branch-&-bound procedure to this instance results in following branch-&-bound tree.*

– Xét trường hợp được mô tả trong Ví dụ 7.1.1. Đồ thị ban đầu chỉ chứa các cung liên hợp & được mô tả trong Hình 7.3a: Đồ thị thứ tự ưu tiên ở Cấp độ 1 trong Ví dụ 7.1.4. *Makespan tương ứng với đồ thị này là 22. Áp dụng thủ tục branch-&-bound cho trường hợp này sẽ cho ra cây branch-&-bound sau.*

p. 194+++

- 7.2. Shifting Bottleneck Heuristic & Makespan.

- 8. Open Shops (Deterministic).

PART II: STOCHASTIC MODELS.

- 9. Stochastic Models: Preliminaries. Production environments in real world are subject to many sources of uncertainty or randomness. Sources of uncertainty that may have a major impact include machine breakdowns & unexpected releases of high priority jobs. Another source of uncertainty lies in processing times, which are often not precisely known in advance. A good model for a scheduling problem should address these forms of uncertainty.

  – Môi trường sản xuất trong thế giới thực chịu ảnh hưởng của nhiều yếu tố bất định hoặc ngẫu nhiên. Những yếu tố bất định có thể gây ra tác động lớn bao gồm sự cố máy móc & việc phát hành bất ngờ các công việc ưu tiên cao. 1 yếu tố bất định khác nằm ở thời gian xử lý, thường không được biết trước chính xác. 1 mô hình tốt cho bài toán lập lịch trình nên giải quyết được những yếu tố bất định này.

  There are several ways in which such forms of randomness can be modeled. E.g., one could model possibility of machine breakdowns as an integral part of processing times. This can be done by modifying distribution of processing times to take into account possibility of breakdowns. Alternatively, one may model breakdowns as a separate stochastic process, that determines when a machine is available & when it is not.

  – Có 1 số cách để mô hình hóa các dạng ngẫu nhiên như vậy. Ví dụ, người ta có thể mô hình hóa khả năng hỏng hóc máy móc như 1 phần không thể thiếu của thời gian xử lý. Điều này có thể được thực hiện bằng cách sửa đổi phân phối thời gian xử lý để tính đến khả năng hỏng hóc. Ngoài ra, người ta có thể mô hình hóa sự cố như 1 quy trình ngẫu nhiên riêng biệt, xác định khi nào máy móc khả dụng & khi nào thì không.

  1st sect of this chap describes framework & notation. 2nd sect deals with distributions & classes of distributions. 3rd sect goes over various forms of stochastic dominance. 4th sect discusses effect of randomness on expected value of objective function given a fixed schedule. 5th sect describes several classes of scheduling policies.

  – Phần 1 của chương này mô tả khuôn khổ & ký hiệu. Phần 2 đề cập đến phân phối & các lớp phân phối. Phần 3 đề cập đến các dạng khác nhau của sự thống trị ngẫu nhiên. Phần 4 thảo luận về ảnh hưởng của tính ngẫu nhiên lên giá trị kỳ vọng của hàm mục tiêu cho 1 lịch trình cố định. Phần 5 mô tả 1 số lớp chính sách lập lịch trình.

  - 9.1. Framework & Notation. In what follows, it is assumed: distributions of processing times, release dates & due dates are all known in advance, i.e., at time 0. Actual *outcome* or *realization* of a random processing time only becomes known upon completion of processing; realization of a release data or due date becomes known only at that point in time when it actually occurs.

– Framework & Ký hiệu. Trong phần sau, giả định rằng: phân phối thời gian xử lý, ngày phát hành & ngày đến hạn đều được biết trước, tức là tại thời điểm 0. *kết quả* thực tế hoặc *thực hiện* của thời gian xử lý ngẫu nhiên chỉ được biết khi quá trình xử lý hoàn tất; việc thực hiện dữ liệu phát hành hoặc ngày đến hạn chỉ được biết tại thời điểm thực tế xảy ra.

In this part of book following notation is used. Random variables are capitalized, while actual realized values are in lower case. Job j has following quantities of interest associated with it.

1. X_{ij} = random processing time of job j on machine i; if job j is only to be processed on 1 machine, or if it has same processing times on each 1 of machines it may visit, subscript i is omitted.
2. $\frac{1}{\lambda_{ij}}$ = mean or expected value of random variable X_{ij}.
3. R_j = random release date of job j.
4. D_j = random due date of job j.
5. w_j = weight (or importance factor) of job j.

This notation is not completely analogous to notation used for deterministic scheduling models. Reason why X_{ij} is used as processing time in stochastic scheduling is because of fact that P usually refers to a probability. Weight w_j, similar to that in deterministic models, is basically equivalent to cost of keeping job j in system for 1 unit of time. In queueing theory literature, which is closely related to stochastic scheduling, c_j is often used for weight or cost of job j. c_j, w_j are equivalent.

– Trong phần này của cuốn sách, ký hiệu sau được sử dụng. Các biến ngẫu nhiên được viết hoa, trong khi các giá trị thực tế được viết thường. Công việc j có các số lượng quan tâm sau đây liên quan đến nó.

1. X_{ij} = thời gian xử lý ngẫu nhiên của công việc j trên máy i; nếu công việc j chỉ được xử lý trên 1 máy, hoặc nếu nó có cùng thời gian xử lý trên mỗi 1 trong số các máy mà nó có thể ghé thăm, thì chỉ số dưới i được bỏ qua.
2. $\frac{1}{\lambda_{ij}}$ = giá trị trung bình hoặc kỳ vọng của biến ngẫu nhiên X_{ij}.
3. R_j = ngày phát hành ngẫu nhiên của công việc j.
4. D_j = ngày đến hạn ngẫu nhiên của công việc j.
5. w_j = trọng số (hoặc hệ số tầm quan trọng) của công việc j.

Ký hiệu này không hoàn toàn tương tự với ký hiệu được sử dụng cho các mô hình lập lịch xác định. Lý do tại sao X_{ij} được sử dụng làm thời gian xử lý trong lập lịch ngẫu nhiên là vì P thường đề cập đến 1 xác suất. Trọng số w_j, tương tự như trong các mô hình xác định, về cơ bản tương đương với chi phí duy trì công việc j trong hệ thống trong 1 đơn vị thời gian. Trong các tài liệu về lý thuyết xếp hàng, vốn có liên quan chặt chẽ đến lập lịch ngẫu nhiên, c_j thường được sử dụng để chỉ trọng số hoặc chi phí của công việc j. c_j, w_j là tương đương nhau.

- 9.2. Distributions & Classes of Distributions. Distributions & density functions may take many forms. In what follows, for obvious reasons, only distributions of nonnegative random variables are considered. A density function may be continuous over given intervals & may have mass concentrated at given discrete points. This implies: distribution function may not be differentiable everywhere (see Fig. 9.1: Example of a density function & a distribution function.). In what follows a distribution is made between continuous time distributions & discrete time distributions.

  – Phân phối & Lớp Phân phối. Phân phối & hàm mật độ có thể có nhiều dạng. Trong phần sau, vì những lý do hiển nhiên, chỉ xem xét phân phối của các biến ngẫu nhiên không âm. 1 hàm mật độ có thể liên tục trên các khoảng thời gian cho trước & có thể có khối lượng tập trung tại các điểm rời rạc cho trước. Điều này ngụ ý: hàm phân phối có thể không khả vi ở mọi nơi (xem Hình 9.1: Ví dụ về hàm mật độ & hàm phân phối.). Trong phần sau, 1 phân phối được tạo ra giữa các phân phối thời gian liên tục & phân phối thời gian rời rạc.

  A random variable from a continuous time distribution may assume any real nonnegative value within 1 or more intervals. Distribution function of a continuous time distribution is usually denoted by $F(t)$ & its density function by $f(t)$, i.e.,

$$F(t) = P(X \leq t) = \int_0^t f(t)\,\mathrm{d}t \text{ where } f(t) = \frac{dF(t)}{dt}$$

  provided derivative exists. Furthermore,

$$\overline{F}(t) = 1 - F(t) = P(X \geq t).$$

  – 1 biến ngẫu nhiên từ phân phối thời gian liên tục có thể nhận bất kỳ giá trị thực không âm nào trong vòng 1 hoặc nhiều khoảng. Hàm phân phối của phân phối thời gian liên tục thường được ký hiệu là $F(t)$ & hàm mật độ của nó là $f(t)$, tức là,

$$F(t) = P(X \leq t) = \int_0^t f(t)\,\mathrm{d}t \text{ trong đó } f(t) = \frac{dF(t)}{dt}$$

  với điều kiện đạo hàm tồn tại. Hơn nữa,

$$\overline{F}(t) = 1 - F(t) = P(X \geq t).$$

  An important example of a continuous time distribution is *exponential distribution*. Density function of an exponentially distributed random variable X is

$$f(t) = \lambda e^{-\lambda t},$$

  & corresponding distribution function is

$$F(t) = 1 - e^{-\lambda t},$$

which is equal to probability that X is smaller than t (see Fig. 9.2: Exponential distribution). Mean or expected value of X is

$$E(X) = \int_0^\infty tf(t)\,\mathrm{d}t = \int_0^\infty tdF(t) = \frac{1}{\lambda}.$$

Parameter λ is referred *rate* of exponential distribution.

– 1 ví dụ quan trọng về phân phối liên tục theo thời gian là *phân phối mũ*. Hàm mật độ của biến ngẫu nhiên X phân phối mũ là

$$f(t) = \lambda e^{-\lambda t},$$

& hàm phân phối tương ứng là

$$F(t) = 1 - e^{-\lambda t},$$

bằng với xác suất X nhỏ hơn t (xem Hình 9.2: Phân phối mũ). Giá trị trung bình hoặc giá trị kỳ vọng của X là

$$E(X) = \int_0^\infty tf(t)\,\mathrm{d}t = \int_0^\infty tdF(t) = \frac{1}{\lambda}.$$

Tham số λ được tham chiếu đến *tỷ lệ* của phân phối mũ.

A random variable from a discrete time distribution may assume only values on nonnegative integers, i.e., $P(X = t) \geq 0$ for $t \in \mathbb{N}$ & $P(X = t) = 0$ otherwise. An important discrete time distribution is deterministic distribution. A deterministic random variable assumes a given value with probability 1.

– 1 biến ngẫu nhiên từ phân phối thời gian rời rạc chỉ có thể nhận giá trị trên các số nguyên không âm, tức là $P(X = t) \geq 0$ với $t \in \mathbb{N}$ & $P(X = t) = 0$ nếu không thì. 1 phân phối thời gian rời rạc quan trọng là phân phối xác định. 1 biến ngẫu nhiên xác định nhận 1 giá trị cho trước với xác suất 1.

Another important example of a discrete time distribution is geometric distribution. Probability that a geometrically distributed random variable X assumes value $t \in \mathbb{N}$ is

$$P(X = t) = (1 - q)q^t.$$

Its distribution function is

$$P(X \leq t) = \sum_{s=0}^{t}(1 - q)q^s = 1 - \sum_{s=t+1}^{\infty}(1 - q)q^s = 1 - q^{t+1},$$

& its mean is

$$E(X) = \frac{q}{1 - q}.$$

Completion rate $c(t)$ of a continuous time random variable X with density function $f(t)$ & distribution function $F(t)$ is defined as follows:

$$c(t) = \frac{f(t)}{1 - F(t)}.$$

This completion rate is equivalent to failure rate or hazard rate in reliability theory. For an exponentially distributed random variable $c(t) = \lambda$, $\forall t$. Completion rate is equivalent to failure rate or hazard rate in reliability theory. For an exponentially distributed random variable $c(t) = \lambda$, $\forall t$. Completion rate is independent of t is 1 of reasons why exponential distribution plays an important role in stochastic scheduling. This property is closely related to so-called *memoryless* property of exponential distribution, which implies that distribution of remaining processing time of a job that already has received processing for an amount of time t, is exponentially distributed with rate λ & therefore identical to its processing time distribution at very start of its processing.

– 1 ví dụ quan trọng khác về phân phối thời gian rời rạc là phân phối hình học. Xác suất để 1 biến ngẫu nhiên phân phối hình học X nhận giá trị $t \in \mathbb{N}$ là

$$P(X = t) = (1 - q)q^t.$$

Hàm phân phối của nó là

$$P(X \leq t) = \sum_{s=0}^{t}(1 - q)q^s = 1 - \sum_{s=t+1}^{\infty}(1 - q)q^s = 1 - q^{t+1},$$

& giá trị trung bình của nó là

$$E(X) = \frac{q}{1 - q}.$$

Tỷ lệ hoàn thành $c(t)$ của biến ngẫu nhiên liên tục X với hàm mật độ $f(t)$ & hàm phân phối $F(t)$ được định nghĩa như sau:

$$c(t) = \frac{f(t)}{1 - F(t)}.$$

Tỷ lệ hoàn thành này tương đương với tỷ lệ hỏng hóc hoặc tỷ lệ nguy hiểm trong lý thuyết độ tin cậy. Đối với biến ngẫu nhiên phân phối mũ $c(t) = \lambda$, $\forall t$. Tỷ lệ hoàn thành tương đương với tỷ lệ hỏng hóc hoặc tỷ lệ nguy hiểm trong lý thuyết độ

tin cậy. Đối với biến ngẫu nhiên phân phối mũ $c(t) = \lambda$, $\forall t$. Tỷ lệ hoàn thành độc lập với t là 1 trong những lý do tại sao phân phối mũ đóng vai trò quan trọng trong lập lịch ngẫu nhiên. Thuộc tính này có liên quan chặt chẽ đến cái gọi là thuộc tính *memoryless* của phân phối mũ, ngụ ý rằng phân phối thời gian xử lý còn lại của 1 tác vụ đã được xử lý trong khoảng thời gian t, được phân phối mũ với tỷ lệ λ & do đó giống hệt với phân phối thời gian xử lý của nó tại thời điểm bắt đầu quá trình xử lý.

Completion rate of a discrete time random variable is defined as

$$c(t) = \frac{P(X = t)}{P(X \geq t)}.$$

Discrete time completion rate of geometric distribution is

$$c(t) = \frac{P(X = t)}{P(X \geq t)} = \frac{(1-q)q^t}{q^t} = 1 - q, \ \forall t \in \mathbb{N},$$

which is a constant independent of t. This implies: probability a job is completed at t, given it has not been completed before t, is $1 - q$. So geometric distribution has memoryless property as well. Geometric distribution is, in effect, discrete time counterpart of exponential distribution.

– Tốc độ hoàn thành của 1 biến ngẫu nhiên thời gian rời rạc được định nghĩa là

$$c(t) = \frac{P(X = t)}{P(X \geq t)}.$$

Tốc độ hoàn thành thời gian rời rạc của phân phối hình học là

$$c(t) = \frac{P(X = t)}{P(X \geq t)} = \frac{(1-q)q^t}{q^t} = 1 - q, \ \forall t \in \mathbb{N},$$

là 1 hằng số độc lập với t. Điều này ngụ ý: xác suất 1 công việc được hoàn thành tại t, nếu công việc đó chưa được hoàn thành trước t, là $1 - q$. Vì vậy, phân phối hình học cũng có tính chất không nhớ. Trên thực tế, phân phối hình học là 1 dạng phân phối theo thời gian rời rạc tương ứng với phân phối mũ.

Distributions, either discrete time or continuous time, can be classified based on completion rate. An *Increasing Completion Rate (ICR)* distribution is defined as a distribution whose completion rate $c(t)$ is increasing in t, while a *Decreasing Completion Rate (DCR)* distribution is defined as a distribution whose completion rate is decreasing in t.

– Phân phối, theo thời gian rời rạc hoặc thời gian liên tục, có thể được phân loại dựa trên tỷ lệ hoàn thành. Phân phối *Tỷ lệ Hoàn thành Tăng dần (ICR)* được định nghĩa là phân phối có tỷ lệ hoàn thành $c(t)$ tăng trong t, trong khi phân phối *Tỷ lệ Hoàn thành Giảm dần (DCR)* được định nghĩa là phân phối có tỷ lệ hoàn thành giảm trong t.

A subclass of class of continuous time ICR distributions is class of $Erlang(k, \lambda)$ distributions. $Erlang(k, \lambda)$ distribution is defined as

$$F(t) = 1 - \sum_{r=0}^{k-1} \frac{(\lambda t)^r e^{-\lambda t}}{r!}.$$

$Erlang(k, \lambda)$ is a k-fold convolution of same exponential distribution with rate λ. Mean of $Erlang(k, \lambda)$ distribution is therefore $\frac{k}{\lambda}$. If $k = 1$, then distribution is exponential. If both k, λ go to ∞ while $\frac{k}{\lambda} = 1$, i.e., $k = \lambda$, then $Erlang(k, \lambda)$ approaches deterministic distribution with mean 1. Exponential as well as deterministic distribution are ICR distributions.

– 1 lớp con của lớp phân phối ICR theo thời gian liên tục là lớp phân phối $Erlang(k, \lambda)$. Phân phối $Erlang(k, \lambda)$ được định nghĩa là

$$F(t) = 1 - \sum_{r=0}^{k-1} \frac{(\lambda t)^r e^{-\lambda t}}{r!}.$$

$Erlang(k, \lambda)$ là tích chập k-lần có cùng phân phối mũ với tỷ lệ λ. Do đó, giá trị trung bình của phân phối $Erlang(k, \lambda)$ là $\frac{k}{\lambda}$. Nếu $k = 1$, thì phân phối là mũ. Nếu cả k, λ đều tiến tới ∞ trong khi $\frac{k}{\lambda} = 1$, tức là $k = \lambda$, thì $Erlang(k, \lambda)$ tiến tới phân phối xác định với giá trị trung bình là 1. Phân phối mũ cũng như phân phối xác định là phân phối ICR.

A subclass of class of continuous time DCR distributions is class of *mixtures of exponentials*. A random variable X is distributed according to a mixture of exponentials if it is exponentially distributed with rate λ_j with probability p_j, $j \in [n]$, &

$$\sum_{j=1}^{n} p_j = 1.$$

Exponential distribution is DCR as well as ICR. Class of DCR distributions contains other special distributions. E.g., let X with probability p be exponentially distributed with mean $\frac{1}{p}$ & with probability $1 - p$ be zero. Mean & variance of this distribution are $E(X) = 1, Var(X) = \frac{2}{p} - 1$. When p is very close to 0 this distribution is in what follows referred to as an *Extreme Mixture of Exponentials (EME)* distribution. Of course, similar distributions can be constructed for the discrete time case as well.

– 1 lớp con của lớp phân phối DCR theo thời gian liên tục là lớp *hỗn hợp mũ*. 1 biến ngẫu nhiên X được phân phối theo hỗn hợp mũ nếu nó phân phối mũ với tỷ lệ λ_j với xác suất p_j, $j \in [n]$, &

$$\sum_{j=1}^{n} p_j = 1.$$

Phân phối mũ cũng là DCR như ICR. Lớp phân phối DCR chứa các phân phối đặc biệt khác. Ví dụ, cho X với xác suất p phân phối mũ với trung bình $\frac{1}{p}$ & với xác suất $1-p$ bằng không. Trung bình & phương sai của phân phối này là $E(X) = 1, Var(X) = \frac{2}{p} - 1$. Khi p rất gần 0, phân phối này được gọi là phân phối Hỗn hợp mũ cực đại (EME). Tất nhiên, các phân phối tương tự cũng có thể được xây dựng cho trường hợp thời gian rời rạc.

1 way of measuring variability of a distribution is through its coefficient of variation $C_v(X)$, which is defined as square root of variance (i.e., standard deviation) divided by mean, i.e.,

$$C_v(X) = \frac{\sqrt{Var(X)}}{E(X)} = \frac{\sqrt{E(X^2) - E(X)^2}}{E(X)}.$$

It can be verified easily that $C_v(X)$ of deterministic distribution is 0 & $C_v(X)$ of exponential distribution is 1 Fig. 9.3: Classes of distributions. $C_v(X)$ of an extreme mixture of exponentials may be arbitrary large (it goes to ∞ when p goes to 0). One may expect $C_v(X)$ of geometric to be 1, since geometric is a discrete time counterpart of exponential distribution. However, $C_v(X)$ of geometric is $\frac{1}{\sqrt{q}}$ (Exercise 9.16).

– 1 cách để đo độ biến thiên của 1 phân phối là thông qua hệ số biến thiên $C_v(X)$, được định nghĩa là căn bậc hai của phương sai (tức là độ lệch chuẩn) chia cho giá trị trung bình, tức là,

$$C_v(X) = \frac{\sqrt{Var(X)}}{E(X)} = \frac{\sqrt{E(X^2) - E(X)^2}}{E(X)}.$$

Có thể dễ dàng kiểm chứng rằng $C_v(X)$ của phân phối xác định là 0 & $C_v(X)$ của phân phối mũ là 1 Hình 9.3: Các lớp phân phối. $C_v(X)$ của 1 hỗn hợp cực trị của các phân phối mũ có thể lớn tùy ý (nó tiến tới ∞ khi p tiến tới 0). Ta có thể kỳ vọng $C_v(X)$ của hình học là 1, vì hình học là 1 phân phối rời rạc theo thời gian của phân phối mũ. Tuy nhiên, $C_v(X)$ của hình học là $\frac{1}{\sqrt{q}}$ (Bài tập 9.16).

◦ 9.3. Stochastic Dominance. It occurs often in stochastic scheduling: 2 random variables have to be compared to 1 another. There are many ways in which one can compare random variables to one another. Comparisons are based on properties referred to as *stochastic dominance*, i.e., a random variable *dominates* another w.r.t. some stochastic property. All forms of stochastic dominance presented in this sect apply to continuous random variables as well as to discrete random variables. Discrete time & continuous time definitions are only in a few cases presented separately. Most forms of stochastic dominance can also be applied in comparisons between a continuous random variable & a discrete random variable.

– Sự thống trị ngẫu nhiên. Điều này thường xảy ra trong lập lịch ngẫu nhiên: 2 biến ngẫu nhiên phải được so sánh với 1 biến khác. Có nhiều cách để so sánh các biến ngẫu nhiên với nhau. So sánh dựa trên các đặc tính được gọi là *thống trị ngẫu nhiên*, tức là 1 biến ngẫu nhiên *thống trị* 1 biến ngẫu nhiên khác đối với 1 đặc tính ngẫu nhiên nào đó. Tất cả các dạng thống trị ngẫu nhiên được trình bày trong phần này đều áp dụng cho cả biến ngẫu nhiên liên tục & biến ngẫu nhiên rời rạc. Định nghĩa về thời gian rời rạc & thời gian liên tục chỉ được trình bày riêng trong 1 số trường hợp. Hầu hết các dạng thống trị ngẫu nhiên cũng có thể được áp dụng trong các phép so sánh giữa 1 biến ngẫu nhiên liên tục & 1 biến ngẫu nhiên rời rạc.

Definition 6 (Stochastic dominance based on expectation)**.** *(i) Random variable X_1 is said to be larger in expectation than random variable X_2 if $E(X_1) \geq E(X_2)$.*

(ii) Random variable X_1 is said to be stochastically larger than random variable X_2 if

$$P(X_1 > t) \geq P(X_2 > t), \ \forall t,$$

or

$$1 - F_1(t) \geq 1 - F_2(t), \ \forall t.$$

This ordering is usually referred to as stochastic ordering & is denoted by $X_1 \geq_{st} X_2$.

(iii) Continuous time random variable X_1 is larger than continuous time random variable X_2 in likelihood ratio sense if $\frac{f_1(t)}{f_2(t)}$ is nondecreasing in $t \geq 0$. Discrete time random variable X_1 is larger than discrete time random variable X_2 in likelihood ratio sense if $\frac{P(X_1=t)}{P(X_2=t)}$ is nondecreasing in $t \in \mathbb{N}$. This form of stochastic dominance is denoted by $X_1 \geq_{lr} X_2$.

(iv) Random variable X_1 is almost surely larger than or equal to random variable X_2 if $P(X_1 \geq X_2) = 1$. This ordering implies: density functions f_1, f_2 may overlap at most on 1 point & is denoted by $X_1 \geq_{a.s.} X_2$.

Định nghĩa 5 (Sự thống trị ngẫu nhiên dựa trên kỳ vọng)**.** *(i) Biến ngẫu nhiên X_1 được gọi là có kỳ vọng lớn hơn biến ngẫu nhiên X_2 nếu $E(X_1) \geq E(X_2)$.*

(ii) Biến ngẫu nhiên X_1 được gọi là có kỳ vọng lớn hơn biến ngẫu nhiên X_2 nếu

$$P(X_1 > t) \geq P(X_2 > t),\ \forall t,$$

hoặc

$$1 - F_1(t) \geq 1 - F_2(t),\ \forall t.$$

Thứ tự này thường được gọi là thứ tự ngẫu nhiên & được ký hiệu là $X_1 \geq_{st} X_2$.

(iii) Biến ngẫu nhiên liên tục X_1 lớn hơn biến ngẫu nhiên liên tục X_2 theo nghĩa tỷ lệ khả năng nếu $\frac{f_1(t)}{f_2(t)}$ không giảm trong $t \geq 0$. Biến ngẫu nhiên rời rạc X_1 lớn hơn biến ngẫu nhiên rời rạc X_2 theo nghĩa tỷ lệ khả năng nếu $\frac{P(X_1=t)}{P(X_2=t)}$ không giảm trong $t \in \mathbb{N}$. Dạng trội ngẫu nhiên này được ký hiệu là $X_1 \geq_{lr} X_2$.

(iv) Biến ngẫu nhiên X_1 gần như chắc chắn lớn hơn hoặc bằng biến ngẫu nhiên X_2 nếu $P(X_1 \geq X_2) = 1$. Thứ tự này ngụ ý: các hàm mật độ f_1, f_2 có thể chồng lên nhau nhiều nhất tại 1 điểm & được ký hiệu là $X_1 \geq_{a.s.} X_2$.

Ordering in expectation is crudest form of stochastic dominance. Stochastic ordering implies ordering in expectation since

$$E(X_1) = \int_0^\infty t f_1(t)\,\mathrm{d}t = \int_0^\infty (1 - F_1(t))\,\mathrm{d}t = \int_0^\infty \overline{F}_1(t)\,\mathrm{d}t$$

(see Exercise 9.11). It can easily be shown: likelihood ratio ordering implies stochastic ordering & reverse does not hold.

– Sắp xếp theo kỳ vọng là dạng thô sơ nhất của sự thống trị ngẫu nhiên. Sắp xếp ngẫu nhiên ngụ ý sắp xếp theo kỳ vọng vì

$$E(X_1) = \int_0^\infty t f_1(t)\,\mathrm{d}t = \int_0^\infty (1 - F_1(t))\,\mathrm{d}t = \int_0^\infty \overline{F}_1(t)\,\mathrm{d}t$$

(xem Bài tập 9.11). Có thể dễ dàng chứng minh được: sắp xếp theo tỷ lệ khả năng ngụ ý sắp xếp ngẫu nhiên & điều ngược lại không đúng.

Example 20 (Stochastically ordered random variables)**.** *Consider 2 discrete time random variables X_1, X_2. Both take values on* [3]*. [$P(X_i = j) = k$ list]. X_1, X_2 are stochastically ordered bu not likelihood ratio ordered as ratio $\frac{P(X_1=t)}{P(X_2=t)}$ is no monotone.*

– Xét 2 biến ngẫu nhiên rời rạc theo thời gian X_1, X_2. Cả hai đều lấy giá trị trên [3]*. [$P(X_i = j) = k$ danh sách]. X_1, X_2 được sắp xếp ngẫu nhiên nhưng không được sắp xếp theo tỷ lệ xác suất vì tỷ lệ $\frac{P(X_1=t)}{P(X_2=t)}$ không đơn điệu.*

Also easy to find an example of a pair of random variables that are monotone likelihood ratio ordered but not almost surely ordered.

– Cũng dễ dàng tìm thấy 1 ví dụ về 1 cặp biến ngẫu nhiên có tỷ lệ xác suất đơn điệu được sắp xếp nhưng không được sắp xếp gần như chắc chắn.

p. 258+++

- 9.4. Impact of Randomness on Fixed Schedules. Stochastic ordering $\geq_{st}$ as well as variability ordering $\geq_{cx}$ described in prev sect are restricted versions of another form of dominance known as *increasing convex* ordering.

  – Tác động của tính ngẫu nhiên lên các lịch trình cố định. Thứ tự ngẫu nhiên $\geq_{st}$ cũng như thứ tự biến thiên $\geq_{cx}$ được mô tả trong phần trước là các phiên bản hạn chế của 1 dạng thống trị khác được gọi là thứ tự *lồi tăng dần*.

  Definition 7 (Increasing convex ordering)**.** *A continuous time random variable X_1 is said to be larger than a continuous time random variable X_2 in increasing convex sense if*

$$\int_0^\infty h(t) dF_1(t) \geq \int_0^\infty h(t) dF_2(t)$$

for all increasing convex functions h. Discrete time random variable X_1 is said to be larger than discrete time random variable X_2 in increasing convex sense if

$$\sum_{t=0}^{\infty} h(t) P(X_1 = t) \geq \sum_{t=0}^{\infty} h(t) P(X_2 = t)$$

for all increasing convex functions h. This ordering is denoted by $X_1 \geq_{icx} X_2$.

- 10. Single Machine Models (Stochastic).
- 11. Single Machine Models with Release Dates (Stochastic).
- 12. Parallel Machine Models (Stochastic).

- 13. Flow Shops, Job Shops, & Open Shops (Stochastic). Results for stochastic flow shops, job shops, & open shops are somewhat less extensive than those for their deterministic counterparts.

  – Kết quả cho các cửa hàng dòng ngẫu nhiên, cửa hàng việc làm, & cửa hàng mở có phần kém bao quát hơn so với kết quả cho các đối tác xác định của chúng.

  This chap focuses 1st on nonpreemptive static list policies, i.e., permutation schedules, for stochastic flow shops. Optimal permutation schedules often remain optimal in class of nonpreemptive dynamic policies as well as in class of preemptive dynamic policies. For open shops & job shops, only classes of nonpreemptive dynamic policies & preemptive dynamic policies are considered.

  – Chương này tập trung trước tiên vào các chính sách danh sách tĩnh không ưu tiên, tức là các lịch trình hoán vị, cho các cửa hàng luồng ngẫu nhiên. Các lịch trình hoán vị tối ưu thường vẫn là tối ưu trong cả lớp chính sách động không ưu tiên cũng như trong lớp chính sách động ưu tiên. Đối với các cửa hàng mở & cửa hàng việc làm, chỉ các lớp chính sách động không ưu tiên & chính sách động ưu tiên mới được xem xét.

  Results obtained for stochastic flow shops & job shops are somewhat similar to those obtained for deterministic flow shops & job shops. Stochastic open shops are, however, very different from their deterministic counterparts.

  – Kết quả thu được từ các cửa hàng dòng ngẫu nhiên & cửa hàng việc làm khá giống với kết quả thu được từ các cửa hàng dòng xác định & cửa hàng việc làm. Tuy nhiên, các cửa hàng mở ngẫu nhiên rất khác so với các cửa hàng xác định tương ứng.

  1st sect discusses stochastic flow shops with unlimited intermediate storage & jobs not subject to blocking. 2nd sect deals with stochastic flow shops with 0 intermediate storage; jobs are subject to blocking. 3rd sect focuses on stochastic job shops, & last sect goes over stochastic open shops.

  – Phần 1 thảo luận về các cửa hàng dòng ngẫu nhiên với bộ nhớ trung gian không giới hạn & các công việc không bị chặn. Phần 2 đề cập đến các cửa hàng dòng ngẫu nhiên với 0 bộ nhớ trung gian; các công việc bị chặn. Phần 3 tập trung vào các cửa hàng công việc ngẫu nhiên, & phần cuối cùng thảo luận về các cửa hàng mở ngẫu nhiên.

  - 13.1. Stochastic Flow Shops with Unlimited Intermediate Storage. Consider 2 machines in series with unlimited storage between machines & no blocking. There are n jobs. Processing time of job j on machine 1 is X_{1j}, exponentially distributed with rate λ_j. Processing time of job j on machine 2 is X_{2j}, exponentially distributed with rate μ_j. Objective: find nonpreemptive static list policy or permutation schedule that minimizes expected makespan $E(C_{\max})$.

    – Cửa hàng Dòng ngẫu nhiên với Bộ nhớ trung gian Không giới hạn. Xét 2 máy nối tiếp với bộ nhớ không giới hạn giữa các máy & không bị chặn. Có n công việc. Thời gian xử lý của công việc j trên máy 1 là X_{1j}, phân phối mũ với tốc độ λ_j. Thời gian xử lý của công việc j trên máy 2 là X_{2j}, phân phối mũ với tốc độ μ_j. Mục tiêu: tìm chính sách danh sách tĩnh không ưu tiên hoặc lịch trình hoán vị sao cho tối thiểu hóa khoảng thời gian dự kiến $E(C_{\max})$.

    Note that this problem is a stochastic counterpart of deterministic problem $F2||C_{\max}$. Deterministic 2 machine problem has a very simple solution. It turns out: stochastic version with exponential processing times has a very elegant solution as well.

    Theorem 1. *Sequencing jobs in decreasing order of $\lambda_j - \mu_j$ minimizes expected makespan in class of nonpreemptive static list policies, in class of nonpreemptive dynamic policies, & in class of preemptive dynamic policies.*

    – Việc sắp xếp các tác vụ theo thứ tự giảm dần của $\lambda_j - \mu_j$ sẽ giảm thiểu thời gian thực hiện dự kiến trong lớp chính sách danh sách tĩnh không ưu tiên, trong lớp chính sách động không ưu tiên & trong lớp chính sách động ưu tiên.

    From statement of theorem: number of optimal schedules in exponential case is often smaller than number of optimal schedules in deterministic case.

    – Từ phát biểu định lý: số lượng lịch trình tối ưu trong trường hợp hàm mũ thường nhỏ hơn số lượng lịch trình tối ưu trong trường hợp xác định.

    p. 360+++

  PART III: SCHEDULING IN PRACTICE.

- 14. General Purpose Procedures for Deterministic Scheduling.
- 15. More Advanced General Purpose Procedures.
- 16. Modeling & Solving Scheduling Problems in Practice.
- 17. Design & Implementation of Scheduling Systems: Basic Concepts.
- 18. Design & Implementation of Scheduling Systems: More Advanced Concepts.
- 19. Examples of System Designs & Implementations.
- 20. What Lies Ahead?
- Appendix A: Mathematical Programming: Formulations & Applications.
- Appendix B: Deterministic & Stochastic Dynamic Programming.
- Appendix C: Constraint Programming.

- Appendix D: Complexity Theory.
- Appendix E: Complexity Classification of Deterministic Scheduling Problems.
- Appendix F: Overview of Stochastic Scheduling Problems.
- Appendix G: Selected Scheduling Systems.
- Appendix H: Lekin System.

4 Miscellaneous

Tài liệu

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer-Verlag, Berlin, 2004, pp. xx+546. ISBN: 3-540-40286-1. DOI: 10.1007/978-3-540-24777-7. URL: https://doi.org/10.1007/978-3-540-24777-7.

[NR21] Peter Norvig and Stuart Russell. *Artificial Intelligence: A Modern Approach*. 4th Edition, Global Edition. Pearson Series In Artificial Intelligence. Pearson, 2021, p. 1166.

[Pin22] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 6th edition. Springer, 2022, pp. xvii+698.