Reinforcement Learning – Học Tăng Cường

Nguyễn Quản Bá Hồng*

Ngày 23 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:

- *Reinforcement Learning – Học Tăng Cường.*
  PDF: URL: .pdf.
  TEX: URL: .tex.
- .
  PDF: URL: .pdf.
  TEX: URL: .tex.

Mục lục

1 Basic Reinforcement Learning

1.1 Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. 2e. 2020

- Preface to 2e. 20 years since publication of 1e of this book have seen tremendous progress in AI, propelled in large part by advances in ML, including advances in RL. Although impressive computational power that became available is responsible for some of these advances, new developments in theory & algorithms have been driving forces as well. In face of this progress, a 2e of 1998 book was long overdue, & finally began project in 2012. Goal for 2e was same as goal for 1e: provide a clear & simple account of key ideas & algorithms of RL that is accessible to readers in all related disciplines. Edition remains an introduction, & retain a focus on core, online learning algorithms. This edition includes some new topics that rose to importance over intervening years, & expanded coverage of topics that we now understand better. But made no attempt to provide comprehensive coverage of field, which has exploded in many different directions. Apologize for having to leave out all but a handful of these contributions.

  – 20 năm kể từ khi xuất bản ấn bản đầu tiên của cuốn sách này, chúng ta đã chứng kiến những tiến bộ vượt bậc trong lĩnh vực AI, phần lớn nhờ vào những tiến bộ trong Học máy (ML), bao gồm cả những tiến bộ trong Học máy (RL). Mặc dù sức mạnh tính toán ấn tượng đã góp phần tạo nên 1 số tiến bộ này, nhưng những phát triển mới về lý thuyết & thuật toán cũng là động lực thúc đẩy. Trước những tiến bộ này, ấn bản thứ 2 của cuốn sách năm 1998 đã bị trì hoãn từ lâu, & cuối cùng đã được khởi động dự án vào năm 2012. Mục tiêu của ấn bản thứ 2 cũng giống như mục tiêu của ấn bản thứ 1: cung cấp 1 bản tóm tắt rõ ràng & đơn giản về những ý tưởng chính & thuật toán của Học máy (RL), dễ hiểu cho độc giả trong mọi lĩnh vực liên quan. Ấn bản này vẫn là phần giới thiệu, & tập trung vào các thuật toán học tập trực tuyến cốt lõi. Ấn bản này bao gồm 1 số chủ đề mới đã trở nên quan trọng trong những năm tiếp theo, & mở rộng phạm vi bao quát các chủ đề mà giờ đây chúng ta đã hiểu rõ hơn. Tuy nhiên, ấn bản này không cố gắng cung cấp phạm vi bao quát toàn diện về lĩnh vực này, vốn đã bùng nổ theo nhiều hướng khác nhau. Xin lỗi vì đã phải bỏ qua hầu hết những đóng góp này.

*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.

As in 1e, chose not to produce a rigorous formal treatment of RL, or to formulate it in most general terms. However, our deeper understanding of some topics since 1e required a bit more mathematics to explain; have set off more mathematical parts in shaded boxes that non-mathematically-inclined may choose to skip. Also use a slightly different notation used in 1e. In teaching, have found: new notation helps to address some common points of confusion. It emphasizes difference between random variables, denoted with capital letters, & their instantiations, denoted in lower case. E.g., state, action, & reward at time step t are denoted S_t, A_t, R_t, while their possible values might be denoted s, a, r. Along with this, natural to use lower case for value functions (e.g., v_π) & restrict capitals to their tabular estimates (e.g., $Q_t(s, a)$). Approximate value functions are deterministic functions of random parameters & are thus also in lower case (e.g., $\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$). Vectors, e.g. weight vector $\mathbf{w}_t$ (formerly $\boldsymbol{\theta}_t$) & feature vector $\mathbf{x}_t$ (formerly $\boldsymbol{\phi}_t$), are bold & written in lowercase even if they are random variables. Uppercase bold is reserved for matrices. In 1e used special notations, $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$, for transition probabilities & expected rewards. 1 weakness of that notation: it still did not fully characterize dynamics of rewards, giving only their expectations, which is sufficient for dynamic programming but not for RL. Another weakness is excess of subscripts & superscripts. In this edition, use explicit notation of $p(s', r|s, a)$ for joint probability for next state & reward given current state & action. All changes in notation are summarized in a table on p. 6.

– Giống như trong 1e, chúng tôi đã chọn không đưa ra 1 cách xử lý chính thức nghiêm ngặt về RL, hoặc xây dựng nó theo những thuật ngữ chung nhất. Tuy nhiên, sự hiểu biết sâu sắc hơn của chúng tôi về 1 số chủ đề kể từ 1e đòi hỏi nhiều kiến thức toán học hơn để giải thích; đã đặt nhiều phần toán học hơn trong các ô tô đậm mà những người không có thiên hướng toán học có thể chọn bỏ qua. Ngoài ra, chúng tôi cũng sử dụng 1 ký hiệu hơi khác được sử dụng trong 1e. Trong giảng dạy, chúng tôi đã tìm thấy: ký hiệu mới giúp giải quyết 1 số điểm dễ nhầm lẫn phổ biến. Nó nhấn mạnh sự khác biệt giữa các biến ngẫu nhiên, được ký hiệu bằng chữ in hoa, & các thể hiện của chúng, được ký hiệu bằng chữ thường. Ví dụ: trạng thái, hành động, & phần thưởng tại bước thời gian t được ký hiệu là S_t, A_t, R_t, trong khi các giá trị khả dĩ của chúng có thể được ký hiệu là s, a, r. Cùng với điều này, việc sử dụng chữ thường cho các hàm giá trị (ví dụ: v_π) & giới hạn chữ hoa trong các ước lượng dạng bảng của chúng là điều tự nhiên (ví dụ: $Q_t(s, a)$). Hàm giá trị gần đúng là hàm xác định của các tham số ngẫu nhiên & do đó cũng được viết thường (ví dụ: $\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$). Các vectơ, ví dụ: vectơ trọng số $\mathbf{w}_t$ (trước đây là $\boldsymbol{\theta}_t$) & vectơ đặc trưng $\mathbf{x}_t$ (trước đây là $\boldsymbol{\phi}_t$), được viết đậm & bằng chữ thường ngay cả khi chúng là các biến ngẫu nhiên. Chữ in hoa đậm được dành riêng cho các ma trận. Trong 1e đã sử dụng các ký hiệu đặc biệt, $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$, cho xác suất chuyển đổi & phần thưởng mong đợi. 1 điểm yếu của ký hiệu đó: nó vẫn không mô tả đầy đủ động lực của phần thưởng, chỉ đưa ra kỳ vọng của chúng, đủ cho lập trình động nhưng không đủ cho RL. Một điểm yếu khác là có quá nhiều chỉ số dưới & chỉ số trên. Trong phiên bản này, hãy sử dụng ký hiệu rõ ràng $p(s', r|s, a)$ cho xác suất kết hợp cho trạng thái tiếp theo & phần thưởng dựa trên trạng thái hiện tại & hành động. Tất cả các thay đổi về ký hiệu được tóm tắt trong bảng ở trang 6.

2e is significantly expanded, & its top-level organization has been changed. After introductory 1st chap, 2e is divided into 3 new parts. 1st part (Chaps. 2–8) treats as much of RL as possible without going beyond tabular case for which exact solutions can be found. Cover both learning & planning methods for tabular case, as well as their unification in n-step methods & in Dyna. Many algorithms presented in this part are new to 2e, including UCB, Expected Sarsa, Double learning, tree-backup, $Q(\sigma)$, RTDP, & MCTS. Doing tabular case 1st, & thoroughly, enables core ideas to be developed in simplest possible setting. 2nd part of book (Chaps. 9–13) is then devoted to extending ideas to function approximation. It has new sects on ANNs, Fourier basis, LSTD, kernel-based methods, Gradient-TD & Emphatic-TD methods, average-reward methods, true online TD(λ), & policy-gardient methods. 2e significantly expands treatment of off-policy learning, 1st for tabular case in Chaps. 5–7, then with function approximation in Chaps. 11–12. Another change: 2e separates forward-view idea of n-step boostrapping (now treated more fully in Chap. 7) from backward-view idea of eligibility traces (now treated independently in Chap. 12). 3rd part of book has large new chaps on RL's relationships to psychology (Chap. 14) & neuroscience (Chap. 15), as well as an updated case-studies chap including Atari game playing, Watson's wagering strategy, & Go playing programs AlphaGo & AlphaGo Zero (Chap. 16). Still, out of necessity we have included only a small subset of all that has been done in field. Our choices reflect our long-standing interests in inexpensive model-free methods that should scale well to large applications. Final chap now includes a discussion of future societal impacts of RL. For better or worst, 2e is about twice as large as 1e.

– 2e is significantly expanded, & its top-level organization has been changed. After introductory 1st chap, 2e is divided into 3 new parts. 1st part (Chaps. 2–8) treats as much of RL as possible without going beyond tabular case for which exact solutions can be found. Cover both learning & planning methods for tabular case, as well as their unification in n-step methods & in Dyna. Many algorithms presented in this part are new to 2e, including UCB, Expected Sarsa, Double learning, tree-backup, $Q(\sigma)$, RTDP, & MCTS. Doing tabular case 1st, & thoroughly, enables core ideas to be developed in simplest possible setting. 2nd part of book (Chaps. 9–13) is then devoted to extending ideas to function approximation. It has new sects on ANNs, Fourier basis, LSTD, kernel-based methods, Gradient-TD & Emphatic-TD methods, average-reward methods, true online TD(λ), & policy-gardient methods. 2e significantly expands treatment of off-policy learning, 1st for tabular case in Chaps. 5–7, then with function approximation in Chaps. 11–12. Another change: 2e separates forward-view idea of n-step boostrapping (now treated more fully in Chap. 7) from backward-view idea of eligibility traces (now treated independently in Chap. 12). 3rd part of book has large new chaps on RL's relationships to psychology (Chap. 14) & neuroscience (Chap. 15), as well as an updated case-studies chap including Atari game playing, Watson's wagering strategy, & Go playing programs AlphaGo & AlphaGo Zero (Chap. 16). Still, out of necessity we have included only a small subset of all that has been done in field. Our choices reflect our long-standing interests in inexpensive model-free methods that should scale well to large applications. Final chap now includes a discussion of future societal impacts of RL. For better or worst, 2e is about twice as large as 1e.

This book is designed to be used as primary text for a 1- or 2-semester course on RL. For a 1-semester course, 1st 10 chaps should be covered in order & form a good core, to which can be added material from other chaps, from other books e.g. Bertsekas & Tsitsiklis (1996), Wiering & van Otterlo (2012), & Szepesvári (2010), or from literature, according to taste.

Depending of students' background, some additional material on online supervised learning may be helpful. Ideas of options & option models are a natural addition (Sutton, Precup & Singh, 1999). A 2-semester course can cover all chaps as well as supplementary material. Book can also be used as part of broader courses on Ml, AI, or neural networks. In this case, it may be desirable to cover only a subset of material. Recommend covering Chap. 1 for a brief overview, Chap. 2 through Sect. 2.4, Chap. 3, & then selecting sections from remaining chaps according to time & interests. Chap. 6 is most important for subset & for rest of book. A course focusing on ML or neural networks should cover Chaps. 9–10, & a course focusing on AI or planning should cover Chap. 8. Throughout book, sects & chaps that are more difficult & not essential to rest of book are marked with a *. These can be omitted on 1st reading without creating problems later on. Some exercises are also marked with a * to indicate: they are more advanced & not essential to understanding basic material of chap.

– Cuốn sách này được thiết kế để sử dụng làm tài liệu chính cho khóa học 1 hoặc 2 học kỳ về RL. Đối với khóa học 1 học kỳ, 10 chương đầu tiên nên được học theo thứ tự & tạo thành 1 cốt lõi tốt, có thể thêm tài liệu từ các chương khác, từ các cuốn sách khác, ví dụ như Bertsekas & Tsitsiklis (1996), Wiering & van Otterlo (2012) & Szepesvári (2010) hoặc từ tài liệu tham khảo, tùy theo sở thích. Tùy thuộc vào nền tảng của sinh viên, 1 số tài liệu bổ sung về học tập có giám sát trực tuyến có thể hữu ích. Ý tưởng về các tùy chọn & mô hình tùy chọn là 1 sự bổ sung tự nhiên (Sutton, Precup & Singh, 1999). Một khóa học 2 học kỳ có thể bao gồm tất cả các chương cũng như tài liệu bổ sung. Sách cũng có thể được sử dụng như 1 phần của các khóa học rộng hơn về ML, AI hoặc mạng nơ-ron. Trong trường hợp này, có thể chỉ nên học 1 phần nhỏ tài liệu. Đề xuất học Chương 1 để có cái nhìn tổng quan ngắn gọn, Chương Từ Chương 2 đến Chương 2.4, Chương 3, & sau đó chọn các phần từ các chương còn lại theo thời gian & sở thích. Chương 6 là quan trọng nhất đối với tập con & cho phần còn lại của sách. Một khóa học tập trung vào Học máy hoặc mạng nơ-ron nên bao gồm các Chương 9–10, & 1 khóa học tập trung vào Trí tuệ nhân tạo hoặc Lập kế hoạch nên bao gồm Chương 8. Xuyên suốt sách, các chương & khó hơn & không cần thiết cho phần còn lại của sách được đánh dấu bằng dấu *. Những chương này có thể được bỏ qua khi đọc lần đầu mà không gây ra vấn đề gì sau này. Một số bài tập cũng được đánh dấu bằng dấu * để chỉ ra: chúng nâng cao hơn & không cần thiết cho việc hiểu nội dung cơ bản của chương.

- Preface to 1e. 1st came to focus on what is now known as RL in late 1979. Both at University of Massachusetts, work on 1 of earliest projects to revive idea that networks of neuronlike adaptive elements might prove to be a promising approach to artificial adaptive intelligence. Project explored "heterostatic theory of adaptive systems" developed by A. Harry Klopf. Harry's work was a rich source of ideas, & we were permitted to explore them critically & compare them with long history of prior work in adaptive systems. Our task become 1 of teasing ideas apart & understanding their relationships & relative importance. This continues today, but in 1979 we came to realize: perhaps simplest of ideas, which had long been taken for granted, had received surprisingly little attention from a computational perspective. This was simply idea of a learning system that *wants* something, that adapts its behavior in order to maximize a special signal from its environment. This was idea of a "hedonistic" learning system, or, as we would say now, idea of RL.

  – Đầu tiên, chúng tôi tập trung vào cái mà ngày nay được gọi là RL vào cuối năm 1979. Cả hai đều tại Đại học Massachusetts, làm việc trên 1 trong những dự án đầu tiên nhằm khôi phục ý tưởng rằng mạng lưới các yếu tố thích ứng giống nơ-ron có thể là 1 phương pháp tiếp cận đầy hứa hẹn cho trí tuệ nhân tạo thích ứng. Dự án đã khám phá "lý thuyết dị biệt về các hệ thống thích ứng" do A. Harry Klopf phát triển. Công trình của Harry là 1 nguồn ý tưởng phong phú, & chúng tôi được phép khám phá chúng 1 cách phê phán & so sánh chúng với lịch sử lâu dài của các công trình về hệ thống thích ứng trước đây. Nhiệm vụ của chúng tôi là phân tích các ý tưởng & hiểu mối quan hệ & tầm quan trọng tương đối của chúng. Điều này vẫn tiếp tục cho đến ngày nay, nhưng vào năm 1979, chúng tôi nhận ra: có lẽ những ý tưởng đơn giản nhất, vốn từ lâu đã được coi là hiển nhiên, lại nhận được rất ít sự chú ý từ góc độ tính toán. Đây chỉ đơn giản là ý tưởng về 1 hệ thống học tập *nó muốn* 1 thứ gì đó, điều chỉnh hành vi của nó để tối đa hóa 1 tín hiệu đặc biệt từ môi trường của nó. Đây là ý tưởng về 1 hệ thống học tập "hưởng thụ", hay như chúng ta thường nói bây giờ, ý tưởng về RL.

  Like others, had a sense: RL had been thoroughly explored in early days of cybernetics & AI. On closer inspection, though, found: it had been explored only slightly. While RL had clearly motivated some of earliest computational studies of learning, most of these researchers had gone on to other things, e.g. pattern classification, supervised learning, & adaptive control, or they had abandoned study of learning altogether. As a result, special issues involved in learning how to get something from environment received relatively little attention. In retrospect, focusing on this idea was critical step that set this branch of research in motion. Little progress could be made in computational study of RL until it was recognized that such a fundamental idea had not yet been thoroughly explored.

  – Giống như những người khác, tôi có cảm giác: RL đã được khám phá kỹ lưỡng trong những ngày đầu của ngành điều khiển học & AI. Tuy nhiên, khi xem xét kỹ hơn, tôi thấy rằng: nó chỉ được khám phá 1 chút. Trong khi RL rõ ràng đã thúc đẩy 1 số nghiên cứu tính toán sớm nhất về học tập, hầu hết các nhà nghiên cứu này đã chuyển sang những thứ khác, ví dụ như phân loại mẫu, học có giám sát, & điều khiển thích ứng, hoặc họ đã từ bỏ hoàn toàn việc nghiên cứu về học tập. Kết quả là, các vấn đề đặc biệt liên quan đến việc học cách lấy thứ gì đó từ môi trường nhận được tương đối ít sự chú ý. Nhìn lại, việc tập trung vào ý tưởng này là bước quan trọng đưa nhánh nghiên cứu này vào hoạt động. Nghiên cứu tính toán về RL chỉ có thể đạt được rất ít tiến bộ cho đến khi người ta nhận ra rằng 1 ý tưởng cơ bản như vậy vẫn chưa được khám phá kỹ lưỡng.

  Field has come a long way since then, evolving & maturing in several directions. RL has gradually become 1 of most active research areas in ML, AI, & neural network research. Field has developed strong mathematical foundations & impressive applications. Computational study of RL is now a large field, with hundreds of active researchers around world in diverse disciplines e.g. psychology, control theory, AI, & neuroscience. Particularly important have been contributions establishing & developing relationships to theory of optimal control & dynamic programming. Overall problem of learning from interaction to

achieve goals is still far from being solved, but our understanding of it has improved significantly. Can now place component ideas, e.g. temporal-difference learning, dynamic programming, & function approximation, within a coherent perspective w.r.t. overall problem.

– Lĩnh vực này đã đi 1 chặng đường dài kể từ đó, phát triển & trưởng thành theo nhiều hướng. RL đã dần trở thành 1 trong những lĩnh vực nghiên cứu tích cực nhất trong ML, AI, & nghiên cứu mạng nơ-ron. Lĩnh vực này đã phát triển nền tảng toán học vững chắc & các ứng dụng ấn tượng. Nghiên cứu tính toán của RL hiện là 1 lĩnh vực lớn, với hàng trăm nhà nghiên cứu tích cực trên khắp thế giới trong nhiều lĩnh vực khác nhau, ví dụ như tâm lý học, lý thuyết điều khiển, AI, & khoa học thần kinh. Đặc biệt quan trọng là những đóng góp thiết lập & phát triển mối quan hệ với lý thuyết điều khiển tối ưu & lập trình động. Vấn đề chung về học hỏi từ tương tác để đạt được mục tiêu vẫn còn lâu mới được giải quyết, nhưng sự hiểu biết của chúng ta về nó đã được cải thiện đáng kể. Giờ đây, có thể đặt các ý tưởng thành phần, ví dụ như học chênh lệch thời gian, lập trình động, & xấp xỉ hàm, trong 1 quan điểm mạch lạc đối với vấn đề tổng thể.

In some sense we have been working toward this book for 30 years, & have lots of people to thank.

- Summary of notation. Capital letters are used for random variables, whereas lower case letters are used for values of random variable & for scalar functions. Quantities that are required to be real-valued vectors are written in bold & in lower case (even if random variables). Matrices are bold capitals.

- 1. Introduction. Idea that we learn by interacting with our environment is probably the 1st to occur to us when we think about nature of learning. When an infant plays, waves its arms, or looks about, it has not explicit teacher, but it does have gave a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause & effect, about consequences of actions, & about what to do in order to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment & ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, & we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning & intelligence.

  – Ý tưởng rằng chúng ta học bằng cách tương tác với môi trường có lẽ là ý tưởng đầu tiên xuất hiện khi chúng ta nghĩ về bản chất của việc học. Khi 1 đứa trẻ sơ sinh chơi đùa, vẫy tay hoặc nhìn xung quanh, nó không có giáo viên rõ ràng, nhưng nó đã tạo ra 1 kết nối cảm biến vận động trực tiếp với môi trường của mình. Việc thực hiện kết nối này tạo ra rất nhiều thông tin về nguyên nhân & kết quả, về hậu quả của hành động, & về những việc cần làm để đạt được mục tiêu. Trong suốt cuộc đời của chúng ta, những tương tác như vậy chắc chắn là 1 nguồn kiến thức chính về môi trường & bản thân chúng ta. Cho dù chúng ta đang học lái xe hay trò chuyện, chúng ta đều nhận thức sâu sắc về cách môi trường của chúng ta phản ứng với những gì chúng ta làm, & chúng ta tìm cách ảnh hưởng đến những gì xảy ra thông qua hành vi của mình. Học hỏi từ tương tác là 1 ý tưởng nền tảng làm nền tảng cho hầu hết mọi lý thuyết về học tập & trí thông minh.

  In this book, explore a *computational* approach to learning from interaction. Rather than directly theorizing about how people or animals learn, primarily explore idealized learning situations & evaluate effectiveness of various learning methods. I.e., adopt perspective of an AI researcher or engineer. Explore designs for machines that are effective in solving learning problems of scientific or economic interest, evaluating designs through mathematical analysis of computational experiments. Approach we explore, called *reinforcement learning*, is much more focused on goal-directed learning from interaction than other approaches to ML.

  – Trong cuốn sách này, chúng ta sẽ khám phá phương pháp tiếp cận *computation* để học từ tương tác. Thay vì trực tiếp lý thuyết hóa cách con người hoặc động vật học tập, chúng ta sẽ chủ yếu khám phá các tình huống học tập lý tưởng & đánh giá hiệu quả của các phương pháp học tập khác nhau. Ví dụ, hãy áp dụng góc nhìn của 1 nhà nghiên cứu hoặc kỹ sư AI. Khám phá các thiết kế máy móc hiệu quả trong việc giải quyết các vấn đề học tập mang tính khoa học hoặc kinh tế, đánh giá các thiết kế thông qua phân tích toán học các thí nghiệm tính toán. Phương pháp chúng tôi khám phá, được gọi là *reinforcement learning*, tập trung nhiều hơn vào việc học tập hướng mục tiêu từ tương tác so với các phương pháp tiếp cận ML khác.

  - 1.1. RL. RL is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. Learner is not told which actions to take, but instead must discover which actions yield most reward by trying them. In most interesting & challenging cases, actions may affect not only immediate reward but also next situation &, through that, all subsequent rewards. These 2 characteristics – trial-&-error search & delayed reward – are 2 most important distinguishing features of RL.

    – RL đang học cách làm gì – cách liên kết tình huống với hành động – để tối đa hóa tín hiệu phần thưởng số. Người học không được chỉ dẫn phải thực hiện hành động nào, mà thay vào đó, phải tự khám phá hành động nào mang lại phần thưởng cao nhất bằng cách thử nghiệm chúng. Trong hầu hết các trường hợp & thử thách thú vị, hành động có thể ảnh hưởng không chỉ đến phần thưởng tức thời mà còn cả tình huống tiếp theo &, qua đó, tất cả các phần thưởng tiếp theo. Hai đặc điểm này – tìm kiếm thử nghiệm & lỗi & phần thưởng trì hoãn – là 2 đặc điểm phân biệt quan trọng nhất của RL.

    RL, like many topics whose names end with "ing", e.g. ML & mountaineering, is simultaneously a problem, a class of solution methods that work well on problem, & field that studies this problem & its solution methods. Convenient to use a single name for all 3 things, but at same time essential to keep 3 conceptually separate. In particular, distinction between problems & solution methods is very important in RL; failing to make this distinction is source of many confusions.

    – RL, giống như nhiều chủ đề có tên kết thúc bằng "ing", ví dụ: ML & mountaineering, đồng thời là 1 bài toán, 1 lớp các phương pháp giải quyết hiệu quả cho bài toán, & lĩnh vực nghiên cứu bài toán này & các phương pháp giải quyết của nó. Việc sử dụng 1 tên gọi duy nhất cho cả 3 lĩnh vực rất tiện lợi, nhưng đồng thời cũng cần thiết phải tách biệt 3 khái niệm

này về mặt khái niệm. Đặc biệt, việc phân biệt giữa bài toán & phương pháp giải quyết là rất quan trọng trong RL; việc không phân biệt được điều này là nguyên nhân gây ra nhiều nhầm lẫn.

Formalize problem of RL using ideas from dynamical systems theory, specifically, as optimal control of incompletely-known Markov decision processes. Details of this formalization must wait until Chap. 3, but basic idea is simply to capture most important aspects of real problem facing a learning agent interacting over time with its environment to achieve a goal. A learning agent must be able to sense state of its environment to some extent & must be able to take actions that affect state. Agent also must have a goal or goals relating to state of environment. Markov decision processes are intended to include just these 3 aspects – sensation, action, & goal – in their simplest possible forms without trivializing any of them. Any method that is well suited to solving such problems we consider to be RL method.

– Chính thức hóa bài toán RL bằng các ý tưởng từ lý thuyết hệ thống động, cụ thể là điều khiển tối ưu các quá trình quyết định Markov chưa được biết đầy đủ. Chi tiết về chính thức hóa này phải chờ đến Chương 3, nhưng ý tưởng cơ bản chỉ đơn giản là nắm bắt hầu hết các khía cạnh quan trọng của vấn đề thực tế mà 1 tác nhân học tập tương tác theo thời gian với môi trường của nó để đạt được mục tiêu. Một tác nhân học tập phải có khả năng cảm nhận trạng thái của môi trường ở 1 mức độ nào đó & phải có khả năng thực hiện các hành động ảnh hưởng đến trạng thái. Tác nhân cũng phải có 1 hoặc nhiều mục tiêu liên quan đến trạng thái của môi trường. Các quá trình quyết định Markov được thiết kế để chỉ bao gồm 3 khía cạnh này – cảm giác, hành động, & mục tiêu – ở dạng đơn giản nhất có thể mà không tầm thường hóa bất kỳ khía cạnh nào. Bất kỳ phương pháp nào phù hợp để giải quyết các vấn đề như vậy, chúng tôi coi là phương pháp RL.

RL is different from *supervised learning*, kind of learning studied in most current research in field of ML. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification – label – of correct action system should take in that situation, which is often to identify a category to which situation belongs. Object of this kind of learning is for system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in training set. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems, often impractical to obtain examples of desired behavior that are both correct & representative of all situations in which agent has to act. In uncharted territory – where one would expect learning to be most beneficial – an agent must be able to learn from its own experience.

– RL khác với *học có giám sát*, loại học được nghiên cứu trong hầu hết các nghiên cứu hiện tại trong lĩnh vực ML. Học có giám sát là học từ 1 tập huấn luyện các ví dụ được gắn nhãn do 1 người giám sát bên ngoài có hiểu biết cung cấp. Mỗi ví dụ là 1 mô tả về 1 tình huống cùng với 1 thông số kỹ thuật – nhãn – về hành động đúng mà hệ thống nên thực hiện trong tình huống đó, thường là để xác định 1 danh mục mà tình huống đó thuộc về. Mục tiêu của loại học này là để hệ thống suy rộng hoặc khái quát hóa các phản ứng của nó để nó hành động chính xác trong các tình huống không có trong tập huấn luyện. Đây là 1 loại học quan trọng, nhưng chỉ riêng nó thì không đủ để học từ tương tác. Trong các vấn đề tương tác, thường không thực tế để có được các ví dụ về hành vi mong muốn vừa chính xác & đại diện cho tất cả các tình huống mà tác nhân phải hành động. Trong lãnh thổ chưa được khám phá – nơi mà người ta mong đợi việc học có lợi nhất – 1 tác nhân phải có khả năng học hỏi từ kinh nghiệm của chính mình.

RL is also different from what ML researchers call *unsupervised learning*, which is typically about finding structure hidden in collections of unlabeled data. Terms supervised learning & unsupervised learning would seem to exhaustively classify ML paradigms, but they do not. Although one might be tempted to think of RL as a kind of unsupervised learning because it does not rely on examples of correct behavior, RL is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent's experience can certainly be useful in RL, but by itself does not address RL problem of maximizing a reward signal. Therefore consider RL to be a 3rd ML paradigm, alongside supervised learning & unsupervised learning & perhaps other paradigms.

– RL cũng khác với cái mà các nhà nghiên cứu ML gọi là *unsupervised learning*, thường là về việc tìm kiếm cấu trúc ẩn trong các tập hợp dữ liệu chưa được gắn nhãn. Các thuật ngữ học có giám sát & unsupervised learning dường như phân loại đầy đủ các mô hình ML, nhưng chúng không phải vậy. Mặc dù người ta có thể bị cám dỗ nghĩ về RL như 1 loại học không giám sát vì nó không dựa trên các ví dụ về hành vi đúng, RL đang cố gắng tối đa hóa tín hiệu phần thưởng thay vì cố gắng tìm kiếm cấu trúc ẩn. Việc khám phá cấu trúc trong trải nghiệm của 1 tác nhân chắc chắn có thể hữu ích trong RL, nhưng bản thân nó không giải quyết được vấn đề RL về việc tối đa hóa tín hiệu phần thưởng. Do đó, hãy coi RL là mô hình ML thứ 3, cùng với học có giám sát & unsupervised learning & có lẽ là các mô hình khác.

1 of challenges that arise in RL, & not in other kinds of learning, is trade-off between exploration & exploitation. To obtain a lot of reward, a RL agent must prefer actions that it has tried in past & found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. Agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in future. Dilemma: neither exploration nor exploitation can be pursued exclusively without failing at task. Agent must try a variety of action & progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. Exploration–exploitation dilemma has been intensively studied by mathematicians for many decades, yet remains unresolved. For now, simply note: entire issue of balancing exploration & exploitation does not even arise in supervised & unsupervised learning, at least in purest forms of these paradigms.

– Một trong những thách thức nảy sinh trong RL, & không phải trong các loại hình học tập khác, là sự đánh đổi giữa khám phá & khai thác. Để nhận được nhiều phần thưởng, 1 tác nhân RL phải ưu tiên các hành động mà nó đã thử trong quá khứ & thấy hiệu quả trong việc tạo ra phần thưởng. Nhưng để khám phá ra những hành động như vậy, nó phải thử các hành động mà nó chưa từng chọn trước đây. Tác nhân phải *nó khai thác* những gì nó đã trải nghiệm để nhận được phần thưởng, nhưng nó cũng phải *nó khám phá* để đưa ra lựa chọn hành động tốt hơn trong tương lai. Thế tiến thoái lưỡng nan: cả khám phá lẫn khai thác đều không thể được theo đuổi độc quyền mà không thất bại trong nhiệm vụ. Tác nhân phải thử

nhiều hành động & dần dần ưu tiên những hành động có vẻ tốt nhất. Trong 1 nhiệm vụ ngẫu nhiên, mỗi hành động phải được thử nhiều lần để có được ước tính đáng tin cậy về phần thưởng dự kiến của nó. Thế tiến thoái lưỡng nan giữa khám phá & khai thác đã được các nhà toán học nghiên cứu sâu rộng trong nhiều thập kỷ, nhưng vẫn chưa được giải quyết. Hiện tại, chỉ cần lưu ý: toàn bộ vấn đề cân bằng giữa khám phá & khai thác thậm chí không phát sinh trong học tập có giám sát & không giám sát, ít nhất là ở dạng tinh khiết nhất của các mô hình này.

Another key feature of RL: it explicitly considers *whole* problem of a goal-directed agent interacting with an uncertain environment. This is in contrast to many approaches that consider subproblems without addressing how they might fit into a larger picture. E.g., have mentioned: many ML researchers have studied supervised learning without specifying how such an ability would ultimately be useful. Other researchers have developed theories of planning with general goals, but without considering planning's role in real-time decision making, or question of where predictive models necessary for planning would come from. Although these approaches have yielded many useful results, their focus on isolated subproblems is a significant limitation.

– Một đặc điểm quan trọng khác của RL: nó xem xét 1 cách rõ ràng *toàn bộ* vấn đề của 1 tác nhân hướng đến mục tiêu tương tác với 1 môi trường không chắc chắn. Điều này trái ngược với nhiều phương pháp tiếp cận chỉ xem xét các bài toán con mà không giải quyết cách chúng có thể phù hợp với bức tranh tổng thể. Ví dụ, đã đề cập: nhiều nhà nghiên cứu ML đã nghiên cứu học có giám sát mà không chỉ rõ khả năng đó cuối cùng sẽ hữu ích như thế nào. Các nhà nghiên cứu khác đã phát triển các lý thuyết về lập kế hoạch với các mục tiêu chung, nhưng lại không xem xét vai trò của lập kế hoạch trong việc ra quyết định theo thời gian thực, hoặc câu hỏi về việc các mô hình dự đoán cần thiết cho việc lập kế hoạch sẽ đến từ đâu. Mặc dù các phương pháp tiếp cận này đã mang lại nhiều kết quả hữu ích, nhưng việc tập trung vào các bài toán con riêng lẻ là 1 hạn chế đáng kể.

RL takes opposite tack, starting with a complete, interactive, goal-seeking agent. All RL agents have explicit goals, can sense aspects of their environments, & can choose actions to influence their environments. Moreover, usually assumed from beginning: agent has to operate despite significant uncertainty about environment it faces. When RL involves planning, it has to address interplay between planning & real-time action selection, as well as question of how environment models are acquired & improved. When RL involves supervised learning, it does so for specific reasons that determine which capabilities are critical & which are not. For learning research to make progress, important subproblems have to be isolated & studied, but they should be subproblems that play clear roles in complete, interactive, goal-seeking agents, even if all details of complete agent cannot yet be filled in.

– RL có hướng đi ngược lại, bắt đầu với 1 tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu. Tất cả các tác nhân RL đều có mục tiêu rõ ràng, có thể cảm nhận các khía cạnh của môi trường của chúng, & có thể chọn hành động để tác động đến môi trường của chúng. Hơn nữa, thường được giả định ngay từ đầu: tác nhân phải hoạt động bất chấp sự không chắc chắn đáng kể về môi trường mà nó phải đối mặt. Khi RL liên quan đến việc lập kế hoạch, nó phải giải quyết sự tương tác giữa việc lập kế hoạch & lựa chọn hành động theo thời gian thực, cũng như câu hỏi về cách các mô hình môi trường được tiếp thu & cải thiện. Khi RL liên quan đến việc học có giám sát, nó làm như vậy vì những lý do cụ thể để xác định khả năng nào là quan trọng & khả năng nào không. Để nghiên cứu về học tập đạt được tiến bộ, các vấn đề phụ quan trọng phải được phân lập & nghiên cứu, nhưng chúng phải là những vấn đề phụ đóng vai trò rõ ràng trong các tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu, ngay cả khi tất cả các chi tiết của tác nhân hoàn chỉnh vẫn chưa thể được điền đầy đủ.

By a complete, interactive, goal-seeking agent we do not always mean something like a complete organism or robot. These are clearly examples, but a complete, interactive, goal-seeking agent can also be a component of a larger behaving system. In this case, agent directly interacts with rest of larger system & indirectly interacts with larger system's environment. A simple example is an agent that monitors charge level of robot's battery & sends commands to robot's control architecture. This agent's environment is rest of robot together with robot's environment. Important to look beyond most obvious examples of agents & their environments to appreciate generality of RL framework.

– Khi nói đến 1 tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu, chúng ta không phải lúc nào cũng muốn nói đến 1 sinh vật hay robot hoàn chỉnh. Đây rõ ràng là những ví dụ, nhưng 1 tác nhân hoàn chỉnh, tương tác & tìm kiếm mục tiêu cũng có thể là 1 thành phần của 1 hệ thống hoạt động lớn hơn. Trong trường hợp này, tác nhân tương tác trực tiếp với phần còn lại của hệ thống lớn hơn & gián tiếp tương tác với môi trường của hệ thống lớn hơn. Một ví dụ đơn giản là 1 tác nhân theo dõi mức sạc pin của robot & gửi lệnh đến kiến trúc điều khiển của robot. Môi trường của tác nhân này là phần còn lại của robot cùng với môi trường của robot. Điều quan trọng là phải xem xét kỹ hơn hầu hết các ví dụ rõ ràng về tác nhân & môi trường của chúng để đánh giá được tính tổng quát của khuôn khổ RL.

1 of most exciting aspects of modern RL is its substantive & fruitful interactions with other engineering & scientific disciplines. RL is part of a decades-long trend within AI & ML toward greater integration with statistics, optimization, & other mathematical subjects. E.g., ability of some RL methods to learn with parameterized approximators addresses classical "curse of dimensionality" in operations research & control theory. More distinctively, RL has also interacted strongly with psychology & neuroscience, with substantial benefits going both ways. Of all forms of ML, RL is closest to kind of learning that humans & other animals do, & many of core algorithms of RL were originally inspired by biological learning systems. RL has also given back, both through a psychological model of animal learning that better matches some of empirical data, & through an influential model of parts of brain's reward system. Body of this book develops ideas of RL that pertain to engineering & AI, with connections to psychology & neuroscience summarized in Chaps. 14–15.

– Một trong những khía cạnh thú vị nhất của Học máy (RL) hiện đại là những tương tác thiết thực & hiệu quả của nó với các ngành kỹ thuật & khoa học khác. RL là 1 phần của xu hướng kéo dài hàng thập kỷ trong AI & ML hướng tới sự tích hợp sâu hơn với thống kê, tối ưu hóa & các môn toán học khác. Ví dụ, khả năng học với các bộ xấp xỉ tham số của 1 số phương pháp RL giải quyết "lời nguyền đa chiều" cổ điển trong nghiên cứu vận hành & lý thuyết điều khiển. Đặc biệt hơn,

RL cũng tương tác mạnh mẽ với tâm lý học & khoa học thần kinh, mang lại những lợi ích đáng kể cho cả hai bên. Trong tất cả các dạng ML, RL gần nhất với loại hình học tập mà con người & các loài động vật khác thực hiện, & nhiều thuật toán cốt lõi của RL ban đầu được lấy cảm hứng từ các hệ thống học tập sinh học. RL cũng đã đóng góp trở lại, thông qua 1 mô hình tâm lý học về học tập của động vật phù hợp hơn với 1 số dữ liệu thực nghiệm, & thông qua 1 mô hình có ảnh hưởng về các bộ phận của hệ thống khen thưởng của não. Nội dung cuốn sách này phát triển các ý tưởng RL liên quan đến kỹ thuật & AI, với mối liên hệ với tâm lý học & khoa học thần kinh được tóm tắt trong các Chương 14–15.

Finally, RL is also part of a larger trend in AI back toward simple general principles. Since late 1960s, many AI researchers presumed: there are no general principles to be discovered, that intelligence is instead due to possession of a vast number of special purpose tricks, procedures, & heuristics. It was sometimes said: if we could just get enough relevant facts into a machine, say 1 million, or 1 billion, then it would become intelligent. Methods based on general principles, e.g. search or learning, were characterized as "weak methods", whereas those based on specific knowledge were called "strong methods". This view is uncommon today. From our point of view, it was premature: too little effort had been put into search for general principles to conclude that there were none. Modern AI now includes much research looking for general principles of learning, search, & decision making. Not clear how far back pendulum will swing, but RL research is certainly part of swing back toward simpler & fewer general principles of AI.

– Cuối cùng, RL cũng là 1 phần của xu hướng lớn hơn trong AI hướng về các nguyên lý chung đơn giản. Từ cuối những năm 1960, nhiều nhà nghiên cứu AI đã cho rằng: không có nguyên lý chung nào cần được khám phá, mà trí thông minh thực chất là do sở hữu vô số các thủ thuật, quy trình, & phương pháp tìm kiếm chuyên biệt. Đôi khi người ta nói: nếu chúng ta có thể đưa đủ dữ liệu liên quan vào 1 cỗ máy, chẳng hạn như 1 triệu, hoặc 1 tỷ, thì nó sẽ trở nên thông minh. Các phương pháp dựa trên các nguyên lý chung, ví dụ như tìm kiếm hoặc học tập, được gọi là "phương pháp yếu", trong khi các phương pháp dựa trên kiến thức cụ thể được gọi là "phương pháp mạnh". Quan điểm này không phổ biến ngày nay. Theo quan điểm của chúng tôi, điều đó là quá sớm: quá ít nỗ lực được bỏ ra để tìm kiếm các nguyên lý chung để kết luận rằng không có nguyên lý nào cả. AI hiện đại ngày nay bao gồm nhiều nghiên cứu tìm kiếm các nguyên lý chung về học tập, tìm kiếm, & ra quyết định. Không rõ con lắc sẽ quay ngược lại bao xa, nhưng nghiên cứu RL chắc chắn là 1 phần của quá trình quay trở lại với các nguyên lý chung đơn giản hơn & ít nguyên lý chung hơn của AI.

- 1.2. Examples. A good way to understand RL: consider some of examples & possible applications that have guided its development.
  - A master chess player makes a move. Choice is informed both by planning – anticipating possible replies & counterreplies – & by immediate, intuitive judgments of desirability of particular positions & moves.
  - An adaptive controller adjusts parameters of a petroleum refinery's operation in real time. Controller optimizes yield/cost/quality trade-off on basis of specified marginal costs without sticking strictly to set points originally suggested by engineers.
  - A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.
  - A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on current charge level of its battery & how quickly & easily it has been able to find recharger in past.
  - Phil prepares his breakfast. Closely examined, even this apparently mundane activity reveals a complex web of conditional behavior & interlocking goal-subgoal relationships: walking to cupboard, opening it, selecting a cereal box, then reaching for, grasping, & retrieving box. Other complex, tuned, interactive sequences of behavior are required to obtain a bowl, spoon, & milk carton. Each step involves a series of eye movements to obtain information & to guide reaching & locomotion. Rapid judgments are continually made about how to carry objects or whether it is better to ferry some of them to dining table before obtaining others. Each step is guided by goals, e.g. grasping a spoon or getting to refrigerator, & is in service of other goals, e.g. having spoon to eat with once cereal is prepared & ultimately obtaining nourishment. Whether he is aware of it or not, Phil is accessing information about state of his body that determines his nutritional needs, level of hunger, & food preferences.

– Một cách hay để hiểu RL: hãy xem xét 1 số ví dụ & các ứng dụng khả thi đã định hướng cho sự phát triển của nó.

- Một kỳ thủ cờ vua lão luyện thực hiện 1 nước đi. Lựa chọn được đưa ra dựa trên cả việc lập kế hoạch – dự đoán các câu trả lời khả thi & các câu trả lời ngược lại – & bằng những phán đoán trực quan, tức thời về mức độ mong muốn của các vị trí & nước đi cụ thể.
- Một bộ điều khiển thích ứng điều chỉnh các thông số vận hành của 1 nhà máy lọc dầu theo thời gian thực. Bộ điều khiển tối ưu hóa sự cân bằng giữa năng suất/chi phí/chất lượng dựa trên chi phí cận biên cụ thể mà không tuân thủ nghiêm ngặt các điểm đặt ban đầu do các kỹ sư đề xuất.
- Một con linh dương con đang cố gắng đứng dậy sau khi sinh ra vài phút. Nửa giờ sau, nó chạy với tốc độ 20 dặm 1 giờ.
- Một robot di động quyết định liệu nó nên đi vào 1 căn phòng mới để tìm thêm rác để thu gom hay bắt đầu tìm đường quay trở lại trạm sạc pin. Nó đưa ra quyết định dựa trên mức sạc pin hiện tại & tốc độ & dễ dàng mà nó đã tìm thấy bộ sạc trong quá khứ.
- Phil chuẩn bị bữa sáng. Khi xem xét kỹ lưỡng, ngay cả hoạt động tưởng chừng như tầm thường này cũng hé lộ 1 mạng lưới phức tạp của các hành vi có điều kiện & các mối quan hệ mục tiêu-mục tiêu phụ đan xen: đi đến tủ bếp, mở tủ, chọn hộp ngũ cốc, rồi với lấy, nắm lấy, & lấy hộp. Các chuỗi hành vi phức tạp, được điều chỉnh & tương tác khác cũng cần thiết để lấy được bát, thìa, & hộp sữa. Mỗi bước bao gồm 1 loạt chuyển động mắt để thu thập thông tin & hướng dẫn việc với & di chuyển. Những phán đoán nhanh chóng liên tục được đưa ra về cách mang đồ vật hoặc liệu có nên mang 1 số đồ vật đến bàn ăn trước khi lấy những đồ vật khác hay không. Mỗi bước đều được hướng dẫn bởi các mục tiêu, ví dụ:

cầm thìa hay đến tủ lạnh, & phục vụ cho các mục tiêu khác, ví dụ: có thìa để ăn sau khi ngũ cốc đã được chuẩn bị & cuối cùng là có được dinh dưỡng. Dù có nhận thức được điều đó hay không, Phil đang tiếp cận thông tin về trạng thái cơ thể của mình, từ đó quyết định nhu cầu dinh dưỡng, mức độ đói, & sở thích ăn uống của cậu bé.

These examples share features that are so basic that they are easy to overlook. All involve *interaction* between an active decision-making agent & its environment, within which agent seeks to achieve a *goal* despite *uncertainty* about its environment. Agent's actions are permitted to affect future state of environment (e.g., next chess position, level of reservoirs of refinery, robot's next location & future charge level of its battery), thereby affecting actions & opportunities available to agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, & thus may require foresight or planning.

– These examples share features that are so basic that they are easy to overlook. All involve *interaction* between an active decision-making agent & its environment, within which agent seeks to achieve a *goal* despite *uncertainty* about its environment. Agent's actions are permitted to affect future state of environment (e.g., next chess position, level of reservoirs of refinery, robot's next location & future charge level of its battery), thereby affecting actions & opportunities available to agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, & thus may require foresight or planning.

At same time, in all of these examples effects of actions cannot be fully predicted; thus agent must monitor its environment frequently & react appropriately. E.g., Phil must watch milk he pours into his cereal bowl to keep it from overflowing. All these examples involve goals that are explicit in sense: agent can judge progress toward its goal based on what it can sense directly. Chess player knows whether or not he wins, refinery controller knows how much petroleum is being produced, gazelle calf knows when it falls, mobile robot knows when its batteries run down, & Phil knows whether or not he is enjoying his breakfast.

– Đồng thời, trong tất cả các ví dụ này, tác động của hành động không thể được dự đoán đầy đủ; do đó, tác nhân phải thường xuyên theo dõi môi trường xung quanh & phản ứng phù hợp. Ví dụ, Phil phải để ý sữa anh ta đổ vào bát ngũ cốc của mình để tránh bị tràn. Tất cả các ví dụ này đều liên quan đến các mục tiêu rõ ràng: tác nhân có thể đánh giá tiến độ đạt được mục tiêu dựa trên những gì nó có thể cảm nhận trực tiếp. Người chơi cờ vua biết mình thắng hay thua, người quản lý nhà máy lọc dầu biết lượng dầu đang được sản xuất, con linh dương con biết khi nào nó ngã, robot di động biết khi nào pin của nó hết, & Phil biết mình có đang thưởng thức bữa sáng hay không.

In all of these examples agent can use its experience to improve its performance over time. Chess player refines intuition he uses to evaluate positions, thereby improving his play; gazelle calf improves efficiency with which it can run; Phil learns to streamline making his breakfast. Knowledge agent brings to task at start – either from previous experience with related tasks or built into it by design or evolution – influences what is useful or easy to learn, but interaction with environment is essential for adjusting behavior to exploit specific features of task.

– Trong tất cả các ví dụ này, tác nhân có thể sử dụng kinh nghiệm của mình để cải thiện hiệu suất theo thời gian. Người chơi cờ vua tinh chỉnh trực giác mà anh ta sử dụng để đánh giá vị trí, từ đó cải thiện lối chơi của mình; con linh dương Gazelle cải thiện hiệu suất chạy; Phil học cách đơn giản hóa việc chuẩn bị bữa sáng. Kiến thức mà tác nhân mang đến cho nhiệm vụ ngay từ đầu – từ kinh nghiệm trước đó với các nhiệm vụ liên quan hoặc được tích hợp sẵn trong đó theo thiết kế hoặc quá trình tiến hóa – sẽ quyết định những gì hữu ích hoặc dễ học, nhưng tương tác với môi trường là điều cần thiết để điều chỉnh hành vi nhằm khai thác các đặc điểm cụ thể của nhiệm vụ.

- 1.3. Elements of RL. Beyond agent & environment, one can identify 4 main subelements of a RL system: a *policy*, a *reward signal*, a *value function*, &, optionally, a *model* of environment.

  – Ngoài tác nhân & môi trường, người ta có thể xác định 4 yếu tố phụ chính của hệ thống RL: *chính sách*, *tín hiệu phần thưởng*, *hàm giá trị*, &, tùy chọn, *mô hình* của môi trường.

  A *policy* defines learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations. In some cases policy may be a simple function or lookup table, whereas in others it may involve extensive computation e.g. a search process. Policy is core of a RL agent in sense: it alone is sufficient to determine behavior. In general, policies may be stochastic, specifying probabilities for each action.

  – Chính sách xác định cách thức hành xử của tác nhân học tập tại 1 thời điểm nhất định. Nói 1 cách đại khái, chính sách là 1 phép ánh xạ từ các trạng thái nhận thức của môi trường đến các hành động cần thực hiện khi ở trong các trạng thái đó. Nó tương ứng với cái mà trong tâm lý học được gọi là 1 tập hợp các quy tắc hoặc mối liên hệ giữa kích thích & phản ứng. Trong 1 số trường hợp, chính sách có thể là 1 hàm đơn giản hoặc bảng tra cứu, trong khi ở những trường hợp khác, nó có thể liên quan đến các phép tính phức tạp, ví dụ như 1 quy trình tìm kiếm. Chính sách là cốt lõi của 1 tác nhân học tập theo nghĩa: chỉ riêng nó là đủ để xác định hành vi. Nhìn chung, các chính sách có thể là ngẫu nhiên, chỉ định xác suất cho mỗi hành động.

  A *reward signal* defines goal of RL problem. On each time step, environment sends to RL agent a single number called *reward*. Agent's sole objective: maximize total reward it receives over long run. Reward signal thus defines what are good & bad events for agent. in a biological system, might think of rewards as analogous to experiences of pleasure or pain. They are immediate & defining features of problem faced by agent. Reward signal is primary basis for altering policy; if an action selected by policy is followed by low reward, then policy may be changed to select some other action in that situation in future. In general, reward signals may be stochastic functions of state of environment & actions taken.

  – Tín hiệu thưởng xác định mục tiêu của bài toán thực tế ảo (RL). Ở mỗi bước thời gian, môi trường gửi đến tác nhân thực tế ảo 1 số duy nhất gọi là phần thưởng. Mục tiêu duy nhất của tác nhân: tối đa hóa tổng phần thưởng mà nó nhận được

trong thời gian dài. Do đó, tín hiệu thưởng xác định những sự kiện tốt & xấu đối với tác nhân. Trong 1 hệ thống sinh học, có thể coi phần thưởng tương tự như trải nghiệm khoái cảm hoặc đau đớn. Chúng là những đặc điểm & xác định tức thời của vấn đề mà tác nhân gặp phải. Tín hiệu thưởng là cơ sở chính để thay đổi chính sách; nếu 1 hành động được chính sách lựa chọn tiếp theo là phần thưởng thấp, thì chính sách có thể được thay đổi để lựa chọn 1 hành động khác trong tình huống đó trong tương lai. Nhìn chung, tín hiệu thưởng có thể là các hàm ngẫu nhiên của trạng thái môi trường & hành động được thực hiện.

Whereas reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in long run. Roughly speaking, *value* of a state is total amount of reward an agent can expect to accumulate over future, starting from that state. Whereas rewards determine immediate, intrinsic desirability of environmental states, values indicate *long-term* desirability of states after taking into account states that are likely to follow & rewards available in those states. E.g., a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) & pain (if low), whereas values correspond to a more refined & farsighted judgment of how pleased or displeased we are that our environment is in a particular state.

– Trong khi tín hiệu phần thưởng chỉ ra điều gì là tốt theo nghĩa tức thời, thì hàm giá trị chỉ ra điều gì là tốt trong thời gian dài. Nói 1 cách đại khái, *giá trị* của 1 trạng thái là tổng lượng phần thưởng mà 1 tác nhân có thể mong đợi tích lũy trong tương lai, bắt đầu từ trạng thái đó. Trong khi phần thưởng xác định mức độ mong muốn nội tại, tức thời của các trạng thái môi trường, thì các giá trị chỉ ra mức độ mong muốn dài hạn của các trạng thái sau khi tính đến các trạng thái có khả năng theo sau & phần thưởng có sẵn trong các trạng thái đó. Ví dụ: 1 trạng thái có thể luôn mang lại phần thưởng tức thời thấp nhưng vẫn có giá trị cao vì nó thường xuyên được theo sau bởi các trạng thái khác mang lại phần thưởng cao. Hoặc ngược lại cũng có thể đúng. Để đưa ra 1 phép so sánh với con người, phần thưởng có phần giống như niềm vui (nếu cao) & nỗi đau (nếu thấp), trong khi các giá trị tương ứng với 1 & phán đoán tinh tế hơn & có tầm nhìn xa về mức độ chúng ta hài lòng hay không hài lòng khi môi trường của chúng ta ở trong 1 trạng thái cụ thể.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, & only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making & evaluating decisions. Action choices are made based on value judgments. Seek actions that bring about states of highest value, not highest reward, because these actions obtain greatest amount of reward for us over long run. Unfortunately, much harder to determine values than it is to determine rewards. Rewards are basically given directly by environment, but values must be estimated & re-estimated from sequences of observations an agent makes over its entire lifetime. In fact, most important component of almost all RL algorithms we consider is a method for efficiently estimating values. Central role of value estimation is arguably most important thing that has been learned about RL over last 6 decades.

– Phần thưởng theo 1 nghĩa nào đó là chính, trong khi giá trị, như dự đoán về phần thưởng, là thứ yếu. Không có phần thưởng thì không thể có giá trị, & mục đích duy nhất của việc ước tính giá trị là để đạt được nhiều phần thưởng hơn. Tuy nhiên, chính giá trị là thứ chúng ta quan tâm nhất khi đưa ra & đánh giá các quyết định. Các lựa chọn hành động được đưa ra dựa trên các đánh giá về giá trị. Hãy tìm kiếm các hành động mang lại trạng thái có giá trị cao nhất, không phải phần thưởng cao nhất, vì những hành động này mang lại cho chúng ta phần thưởng lớn nhất trong thời gian dài. Thật không may, việc xác định giá trị khó hơn nhiều so với việc xác định phần thưởng. Phần thưởng về cơ bản được trao trực tiếp bởi môi trường, nhưng giá trị phải được ước tính & ước tính lại từ các chuỗi quan sát mà 1 tác nhân thực hiện trong toàn bộ vòng đời của nó. Trên thực tế, thành phần quan trọng nhất của hầu hết các thuật toán RL mà chúng tôi xem xét là 1 phương pháp để ước tính giá trị 1 cách hiệu quả. Vai trò trung tâm của việc ước tính giá trị có thể được cho là điều quan trọng nhất đã được tìm hiểu về RL trong 6 thập kỷ qua.

4th & final element of some RL systems is a *model* of environment. This is something that mimics behavior of environment, or more generally, that allows inferences to be made about how environment will behave. E.g., given a state & action, model might predict resultant next state & next reward. Models are used for *planning*, by which we mean any way of deciding on a course of action by considering possible future situations before they re actually experienced. Methods for solving RL problems that use models & planning are called *model-based* methods, as opposed to simpler *model-free* methods that are explicitly trial-&-error learners – viewed as almost *opposite* of planning. In Chap. 8 explore RL systems that simultaneously learn by trial & error, learn a model of environment, & use model for planning. Modern RL spans spectrum from low-level, trial-&-error learning to high-level, deliberative planning.

– Yếu tố cuối cùng thứ 4 của 1 số hệ thống RL là 1 mô hình môi trường. Đây là thứ mô phỏng hành vi của môi trường, hay nói rộng hơn, cho phép suy luận về cách môi trường sẽ hoạt động. Ví dụ, với 1 trạng thái & hành động, mô hình có thể dự đoán trạng thái tiếp theo & phần thưởng tiếp theo. Các mô hình được sử dụng để lập kế hoạch, tức là bất kỳ cách nào để quyết định 1 hành động bằng cách xem xét các tình huống có thể xảy ra trong tương lai trước khi chúng thực sự được trải nghiệm. Các phương pháp giải quyết các bài toán RL sử dụng mô hình & lập kế hoạch được gọi là các phương pháp dựa trên mô hình, trái ngược với các phương pháp đơn giản hơn là các phương pháp học thử sai rõ ràng – được xem là gần như đối lập với lập kế hoạch. Trong Chương 8, chúng ta sẽ khám phá các hệ thống RL đồng thời học bằng thử sai, học 1 mô hình môi trường & sử dụng mô hình để lập kế hoạch. RL hiện đại trải rộng từ học thử sai cấp thấp đến lập kế hoạch cân nhắc cấp cao.

- 1.4. Limitations & Scope. RL relies heavily on concept of state – as input to policy & value function, & as both input to & output from model. Informally, can think of state as a signal conveying to agent some sense of "how environment is" at a particular time. Formal definition of state as we use it here is given by framework of Markov decision processes presented in Chap. 3. More generally, however, encourage reader to follow informal meaning & think of state as whatever information is available to agent about its environment. In effect, assume: state signal is produced by some preprocessing system that is

nominally part of agent's environment. Do not address issues of constructing, changing, or learning state signal in this book (other than briefly in Sect. 17.3). Take this approach not because we consider state representation to be unimportant, but in order to focus fully on decision-making issues. I.e., our concern in this book is not with designing state signal, but with deciding what action to take as a function of whatever state signal is available.

– RL dựa rất nhiều vào khái niệm trạng thái – là đầu vào cho chính sách & hàm giá trị, & vừa là đầu vào cho & đầu ra từ mô hình. Một cách không chính thức, có thể coi trạng thái là 1 tín hiệu truyền đạt cho tác nhân 1 số cảm giác về "môi trường như thế nào" tại 1 thời điểm cụ thể. Định nghĩa chính thức về trạng thái khi chúng ta sử dụng ở đây được đưa ra bởi khuôn khổ các quy trình quyết định Markov được trình bày trong Chương 3. Tuy nhiên, nói chung hơn, hãy khuyến khích người đọc theo ý nghĩa không chính thức & coi trạng thái là bất kỳ thông tin nào có sẵn cho tác nhân về môi trường của nó. Trên thực tế, hãy giả sử: tín hiệu trạng thái được tạo ra bởi 1 số hệ thống tiền xử lý mà về danh nghĩa là 1 phần của môi trường của tác nhân. Không đề cập đến các vấn đề về xây dựng, thay đổi hoặc học tín hiệu trạng thái trong cuốn sách này (ngoại trừ phần tóm tắt trong Mục 17.3). Sử dụng cách tiếp cận này không phải vì chúng tôi coi biểu diễn trạng thái là không quan trọng, mà là để tập trung hoàn toàn vào các vấn đề ra quyết định. Tức là, mối quan tâm của chúng tôi trong cuốn sách này không phải là thiết kế tín hiệu trạng thái, mà là quyết định hành động nào sẽ thực hiện như 1 hàm của bất kỳ tín hiệu trạng thái nào có sẵn.

Most of RL methods we consider in this book are structured around estimating value functions, but not strictly necessary to do this to solve RL problems. E.g., solution methods e.g. genetic algorithms, genetic programming, simulated annealing, & other optimization methods never estimate value functions. These methods apply multiple static policies each interacting over an extended period of time with a separate instance of environment. Policies that obtain most reward, & random variations of them, are carried over to next generation of policies, & process repeats. Call these *evolutionary* methods because their operation is analogous to way biological evolution produces organisms with skilled behavior even if they do not learn during their individual lifetimes. If space of policies is sufficiently small, or can be structured so that good policies are common or easy to find – or if a lot of time is available for search – then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems on which learning agent cannot sense complete state of its environment.

– Hầu hết các phương pháp RL mà chúng tôi xem xét trong cuốn sách này đều được cấu trúc xung quanh việc ước lượng các hàm giá trị, nhưng không nhất thiết phải làm điều này để giải quyết các bài toán RL. Ví dụ, các phương pháp giải như thuật toán di truyền, lập trình di truyền, ủ mô phỏng, & các phương pháp tối ưu hóa khác không bao giờ ước lượng các hàm giá trị. Các phương pháp này áp dụng nhiều chính sách tĩnh, mỗi chính sách tương tác trong 1 khoảng thời gian dài với 1 môi trường riêng biệt. Các chính sách đạt được nhiều phần thưởng nhất, & các biến thể ngẫu nhiên của chúng, được chuyển sang thế hệ chính sách tiếp theo, & lặp lại quy trình. Gọi chúng là phương pháp *tiến hóa* vì hoạt động của chúng tương tự như cách tiến hóa sinh học tạo ra các sinh vật có hành vi lành nghề ngay cả khi chúng không học trong suốt vòng đời của chúng. Nếu không gian chính sách đủ nhỏ, hoặc có thể được cấu trúc sao cho các chính sách tốt trở nên phổ biến hoặc dễ tìm thấy – hoặc nếu có nhiều thời gian để tìm kiếm – thì phương pháp tiến hóa có thể hiệu quả. Ngoài ra, phương pháp tiến hóa có lợi thế đối với các bài toán mà tác nhân học tập không thể cảm nhận được trạng thái hoàn chỉnh của môi trường.

Our focus is on RL methods that learn while interacting with environment, which evolutionary methods do not do. Methods able to take advantage of details of individual behavioral interactions can be much more efficient than evolutionary methods in many cases. Evolutionary methods ignore much of useful structure of RL problem: they do not use fact: policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases such information can be misleading (e.g., when states are misperceived), but more often it should enable more efficient search. Although evolution & learning share many features & naturally work together, do not consider evolutionary methods by themselves to be especially well suited to RL problems &, accordingly, we do not cover them in this book.

– Trọng tâm của chúng tôi là các phương pháp RL học trong khi tương tác với môi trường, điều mà các phương pháp tiến hóa không làm được. Các phương pháp có thể tận dụng các chi tiết của tương tác hành vi cá nhân có thể hiệu quả hơn nhiều so với các phương pháp tiến hóa trong nhiều trường hợp. Các phương pháp tiến hóa bỏ qua nhiều cấu trúc hữu ích của vấn đề RL: chúng không sử dụng thực tế: chính sách mà chúng đang tìm kiếm là 1 hàm từ trạng thái đến hành động; chúng không nhận thấy trạng thái nào mà 1 cá thể trải qua trong suốt cuộc đời của nó hoặc hành động nào mà nó chọn. Trong 1 số trường hợp, thông tin như vậy có thể gây hiểu lầm (ví dụ: khi các trạng thái bị nhận thức sai), nhưng thông thường hơn, nó sẽ cho phép tìm kiếm hiệu quả hơn. Mặc dù tiến hóa & học tập chia sẻ nhiều đặc điểm & hoạt động tự nhiên cùng nhau, nhưng không coi các phương pháp tiến hóa tự thân là đặc biệt phù hợp với các vấn đề RL &, do đó, chúng tôi không đề cập đến chúng trong cuốn sách này.

- 1.5. An Extended Example: Tic-Tac-Toe. To illustrate general idea of RL & contrast it with other approaches, next consider a single example in more detail. Consider familiar child's game of tic-tac-toe. 2 players take turns playing on a 3-by-3 board. 1 player plays Xs & the other Os until 1 player wins by placing 3 marks in a row, horizontally, vertically, or diagonally, as X player has in game shown to right. If board fills up with neither player getting 3 in a row, then game is a draw. Because a skilled player can play so as never to lose, assume: we are playing against an imperfect player, one whose play is sometimes incorrect & allows us to win. For moment, in fact, consider draws & losses to be equally bad for us. How might we construct a player that will find imperfections in its opponent's play & learn to maximize its chances of winning?

  – Để minh họa ý tưởng chung về RL & so sánh nó với các cách tiếp cận khác, tiếp theo hãy xem xét 1 ví dụ chi tiết hơn. Hãy xem xét trò chơi ô ăn quan quen thuộc của trẻ em. 2 người chơi lần lượt chơi trên 1 bàn cờ 3x3. 1 người chơi đánh X & người kia đánh O cho đến khi 1 người chơi thắng bằng cách đặt 3 điểm liên tiếp, theo chiều ngang, chiều dọc hoặc đường

chéo, như người chơi X đã chỉ ra trong trò chơi ở bên phải. Nếu bàn cờ đầy mà không có người chơi nào đạt được 3 điểm liên tiếp, thì trò chơi hòa. Bởi vì 1 người chơi có kỹ năng có thể chơi để không bao giờ thua, hãy giả sử: chúng ta đang chơi với 1 người chơi không hoàn hảo, người chơi đôi khi chơi không chính xác & cho phép chúng ta thắng. Trên thực tế, hiện tại, hãy coi hòa & thua đều tệ như nhau đối với chúng ta. Làm thế nào chúng ta có thể xây dựng 1 người chơi sẽ tìm ra điểm không hoàn hảo trong cách chơi của đối thủ & học cách tối đa hóa cơ hội chiến thắng của mình?

Although this is a simple problem, it cannot readily be solved in a satisfactory way through classical techniques. E.g., classical "minimax" solution from game theory is not correct here because it assumes a particular way of playing by opponent. E.g., a minimax player would never reach a game state from which it could lose, even if in fact it always won from that state because of incorrect play by opponent. Classical optimization methods for sequential decision problems, e.g. dynamic programming, can compute an optimal solution for any opponent, but require as input a complete specification of that opponent, including probabilities with which opponent makes each move in each board state. Assume: this information is not available a priori for this problem, as it is not for vast majority of problems of practical interest. On other hand, such information can be estimated from experience, in this case by playing many games against opponent. About the best one can do on this problem is 1st to learn a model of opponent's behavior, up to some level of confidence, & then apply dynamic programming to compute an optimal solution given approximate opponent model. In end, this is not that different from some of RL methods we examine later in this book.

– Mặc dù đây là 1 bài toán đơn giản, nhưng nó không thể dễ dàng được giải quyết 1 cách thỏa đáng bằng các kỹ thuật cổ điển. Ví dụ, giải pháp "minimax" cổ điển từ lý thuyết trò chơi không đúng ở đây vì nó giả định 1 cách chơi cụ thể của đối thủ. Ví dụ, 1 người chơi minimax sẽ không bao giờ đạt đến trạng thái trò chơi mà họ có thể thua, ngay cả khi trên thực tế họ luôn thắng từ trạng thái đó do đối thủ chơi không đúng. Các phương pháp tối ưu hóa cổ điển cho các bài toán quyết định tuần tự, ví dụ như lập trình động, có thể tính toán 1 giải pháp tối ưu cho bất kỳ đối thủ nào, nhưng yêu cầu đầu vào là 1 thông số kỹ thuật đầy đủ về đối thủ đó, bao gồm xác suất đối thủ thực hiện mỗi nước đi trong mỗi trạng thái bàn cờ. Giả sử: thông tin này không có sẵn trước cho bài toán này, vì nó không có sẵn cho phần lớn các bài toán thực tế. Mặt khác, thông tin đó có thể được ước tính từ kinh nghiệm, trong trường hợp này là bằng cách chơi nhiều ván với đối thủ. Cách tốt nhất có thể làm cho bài toán này là đầu tiên tìm hiểu 1 mô hình hành vi của đối thủ, với 1 mức độ tin cậy nhất định, & sau đó áp dụng lập trình động để tính toán 1 giải pháp tối ưu dựa trên mô hình đối thủ gần đúng. Cuối cùng, điều này không khác mấy so với 1 số phương pháp RL mà chúng ta sẽ xem xét sau trong cuốn sách này.

An evolutionary method applied to this problem would directly search space of possible policies for one with a high probability of winning against opponent. Here, a policy is a rule that tells player what move to make for every state of game – every possible configuration of Xs & Os on 3×3 board. For each policy considered, an estimate of its winning probability would be obtained by playing some number of games against opponent. This evaluation would then direct which policy or policies were considered next. A typical evolutionary method would hill-climb in policy space, successively generating & evaluating policies in an attempt to obtain incremental improvements. Or, perhaps, a genetic-style algorithm could be used that would maintain & evaluate a population of policies. Literally hundreds of different optimization methods could be applied.

– Một phương pháp tiến hóa được áp dụng cho vấn đề này sẽ trực tiếp tìm kiếm không gian các chính sách khả thi cho 1 chính sách có xác suất chiến thắng cao trước đối thủ. Ở đây, 1 chính sách là 1 quy tắc cho người chơi biết phải thực hiện nước đi nào cho mọi trạng thái của trò chơi – mọi cấu hình có thể có của Xs & O trên bàn cờ 3×3. Đối với mỗi chính sách được xem xét, ước tính xác suất chiến thắng của chính sách đó sẽ được thu được bằng cách chơi 1 số ván đấu nhất định với đối thủ. Đánh giá này sau đó sẽ chỉ ra chính sách hoặc các chính sách nào được xem xét tiếp theo. Một phương pháp tiến hóa điển hình sẽ leo đồi trong không gian chính sách, liên tục tạo ra & đánh giá các chính sách nhằm cố gắng đạt được những cải tiến gia tăng. Hoặc, có lẽ, 1 thuật toán kiểu di truyền có thể được sử dụng để duy trì & đánh giá 1 quần thể các chính sách. Hàng trăm phương pháp tối ưu hóa khác nhau có thể được áp dụng.

Here is how tic-tac-toe problem would be approached with a method making use of a value function. 1st would set up a table of numbers, one for each possible state of game. Each number will be latest estimate of probability of our winning from that state. Treat this estimate as state's *value*, & whole table is learned value function. State A is higher value than state B, or is considered "better" than state B, if current estimate of probability of our winning from A is higher than it is from B. Assuming we always play Xs, then for all states with 3 Xs in a row probability of winning is 1, because we have already won. Similarly, for all states with 3 Os in a row, or that are filled up, correct probability is 0, as we cannot win from them. Set initial values of all other states to 0.5, representing a guess that we have a 50% chance of winning.

– Sau đây là cách tiếp cận bài toán ô ăn quan bằng phương pháp sử dụng hàm giá trị. Đầu tiên, hãy lập 1 bảng số, mỗi số cho 1 trạng thái có thể xảy ra của trò chơi. Mỗi số sẽ là ước tính mới nhất về xác suất chiến thắng của chúng ta từ trạng thái đó. Hãy coi ước tính này là *giá trị của nó* của trạng thái, & toàn bộ bảng là hàm giá trị đã học. Trạng thái A có giá trị cao hơn trạng thái B, hoặc được coi là "tốt hơn" trạng thái B, nếu ước tính hiện tại về xác suất chiến thắng của chúng ta từ A cao hơn từ B. Giả sử chúng ta luôn chơi X, thì đối với tất cả các trạng thái có 3 X liên tiếp, xác suất chiến thắng là 1, vì chúng ta đã thắng. Tương tự, đối với tất cả các trạng thái có 3 O liên tiếp, hoặc đã được lấp đầy, xác suất đúng là 0, vì chúng ta không thể thắng từ chúng. Đặt giá trị ban đầu của tất cả các trạng thái khác thành 0,5, thể hiện dự đoán rằng chúng ta có 50% cơ hội chiến thắng.

Then play many games against opponent. To select our moves, examine states that would result from each of our possible moves (one for each blank space on board) & look up their current values in table. Most of time, we move *greedily*, selecting move that leads to state with greatest value, i.e., with highest estimated probability of winning. Occasionally, however, select randomly from among the other moves instead. These are called *exploratory* moves because they cause us to experience states that we might otherwise never see. A sequence of moves made & considered during a game can be diagrammed as in

Fig. 1.1: A sequence of tic-tac-toe moves. Solid black lines represent moves taken during a game; dashed lines represent moves

that we (our RL player) considered but did not make. * indicates move currently estimated to be the best. Our 2nd move was an exploratory move, i.e., it was taken even though another sibling move, the one leading to e^*, was ranked higher. Exploratory moves do not result in any learning, but each of our other moves does, causing updates are suggested by red arrows in which estimated values are moved up tree from later nodes to earlier nodes as detailed in text.

– Sau đó chơi nhiều ván đấu với đối thủ. Để chọn nước đi, hãy kiểm tra các trạng thái có thể xảy ra từ mỗi nước đi khả thi của chúng ta (một nước đi cho mỗi ô trống trên bàn cờ) & tra cứu giá trị hiện tại của chúng trong bảng. Hầu hết thời gian, chúng ta di chuyển *1 cách tham lam*, chọn nước đi dẫn đến trạng thái có giá trị lớn nhất, tức là có xác suất chiến thắng ước tính cao nhất. Tuy nhiên, đôi khi, chúng ta chọn ngẫu nhiên từ các nước đi khác. Đây được gọi là nước đi *khám phá* vì chúng khiến chúng ta trải nghiệm các trạng thái mà nếu không thì chúng ta có thể sẽ không bao giờ thấy. Một chuỗi các nước đi được thực hiện & được xem xét trong 1 ván cờ có thể được biểu diễn như trong Hình 1.1: Một chuỗi các nước đi ô ăn quan. Các đường liền màu đen biểu diễn các nước đi được thực hiện trong 1 ván cờ; các đường đứt nét biểu diễn các nước đi mà chúng ta (người chơi thực tế của chúng ta) đã cân nhắc nhưng không thực hiện. * biểu thị nước đi hiện được ước tính là tốt nhất. Nước đi thứ 2 của chúng ta là 1 nước đi khám phá, tức là nó được thực hiện mặc dù 1 nước đi khác cùng nhóm, nước đi dẫn đến e^*, được xếp hạng cao hơn. Các bước di chuyển khám phá không mang lại bất kỳ bài học nào, nhưng mỗi bước di chuyển khác của chúng ta đều mang lại, khiến các bản cập nhật được gợi ý bằng các mũi tên màu đỏ trong đó các giá trị ước tính được di chuyển lên cây từ các nút sau đến các nút trước như được trình bày chi tiết trong văn bản.

While we are playing, change values of states in which we find ourselves during game. Attempt to make them more accurate estimates of probabilities of winning. To do this, "back up" value of state after each greedy move to state before move, as suggested by arrows in Fig. 1.1. More precisely, current value of earlier state is updated to be closer to value of later state. This can be done by moving earlier state's value a fraction of way toward value of later state. If let S_t denote state before greedy move, & S_{t+1} state after that move, then update to estimated value of S_t, denoted $V(S_t)$, can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)],$$

where α is a small positive fraction called *step-size parameter*, which influences rate of learning. This update rule is an example of a *temporal-difference* learning method, so called because its changes are based on a difference, $V(S_{t+1}) - V(S_t)$, between estimates at 2 successive times.

– Trong khi chơi, hãy thay đổi giá trị của các trạng thái mà chúng ta đang ở trong trò chơi. Cố gắng ước tính chính xác hơn xác suất chiến thắng. Để làm điều này, hãy "sao lưu" giá trị của trạng thái sau mỗi lần di chuyển tham lam đến trạng thái trước khi di chuyển, như được gợi ý bằng các mũi tên trong Hình 1.1. Chính xác hơn, giá trị hiện tại của trạng thái trước đó được cập nhật để gần hơn với giá trị của trạng thái sau đó. Điều này có thể được thực hiện bằng cách di chuyển giá trị của trạng thái trước đó 1 phần về phía giá trị của trạng thái sau đó. Nếu cho S_t biểu thị trạng thái trước khi di chuyển tham lam, & S_{t+1} trạng thái sau khi di chuyển đó, thì cập nhật thành giá trị ước tính của S_t, ký hiệu là $V(S_t)$, có thể được viết như sau

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)],$$

trong đó α là 1 phân số dương nhỏ được gọi là *tham số kích thước bước*, ảnh hưởng đến tốc độ học. Quy tắc cập nhật này là 1 ví dụ về phương pháp học *chênh lệch thời gian*, được gọi như vậy vì các thay đổi của nó dựa trên chênh lệch $V(S_{t+1}) - V(S_t)$, giữa các ước tính tại 2 thời điểm liên tiếp.

Method described above performs quite well on this task. E.g., if step-size parameter is reduced properly over time, then this method converges, for any fixed opponent, to true probabilities of winning from each state given optimal play by our player. Furthermore, moves then taken (except on exploratory moves) are in fact optimal moves against this (imperfect) opponent. I.e., method converges to an optimal policy for playing game against this opponent. If step-size parameter is not reduced all way to 0 over time, then this player also plays well against opponents that slowly change their way of playing.

– Phương pháp được mô tả ở trên thực hiện khá tốt nhiệm vụ này. Ví dụ, nếu tham số bước nhảy được giảm dần theo thời gian, thì phương pháp này sẽ hội tụ, đối với bất kỳ đối thủ cố định nào, về xác suất chiến thắng thực sự từ mỗi trạng thái với lối chơi tối ưu của người chơi. Hơn nữa, các nước đi được thực hiện sau đó (trừ các nước đi thăm dò) thực chất là các nước đi tối ưu khi đối đầu với đối thủ (không hoàn hảo) này. Tức là, phương pháp này hội tụ về 1 chiến lược tối ưu để chơi với đối thủ này. Nếu tham số bước nhảy không giảm hoàn toàn về 0 theo thời gian, thì người chơi này cũng chơi tốt với những đối thủ dần thay đổi cách chơi.

This example illustrates differences between evolutionary methods & methods that learn value functions. To evaluate a policy, an evolutionary method holds policy fixed & plays many games against opponent or simulates many games using a model of opponent. Frequency of wins gives an unbiased estimate of probability of winning with that policy, & can be used to direct next policy selection. But each policy change is made only after many games, & only final outcome of each game is used: what happens *during* the games is ignored. E.g., if player wins, then *all* of its behavior in game is given credit, independently of how specific moves might have been critical to win. Credit is even given to moves that never occurred! Value function methods, in contrast, allow individual states to be evaluated. In end, evolutionary & value function methods both search space of policies, but learning a value function takes advantage of information available during course of play.

– Ví dụ này minh họa sự khác biệt giữa các phương pháp tiến hóa & các phương pháp học hàm giá trị. Để đánh giá 1 chính sách, 1 phương pháp tiến hóa giữ chính sách cố định & chơi nhiều ván đấu với đối thủ hoặc mô phỏng nhiều ván đấu bằng cách sử dụng mô hình của đối thủ. Tần suất chiến thắng đưa ra ước tính khách quan về xác suất chiến thắng với chính sách đó, & có thể được sử dụng để chỉ đạo lựa chọn chính sách tiếp theo. Nhưng mỗi lần thay đổi chính sách chỉ được thực hiện sau nhiều ván đấu, & chỉ kết quả cuối cùng của mỗi ván đấu được sử dụng: những gì xảy ra *trong* các ván đấu bị bỏ qua. Ví dụ: nếu người chơi thắng, thì *tất cả* hành vi của người chơi trong ván đấu đều được ghi nhận, bất kể các nước đi cụ thể

có thể quan trọng như thế nào để giành chiến thắng. Thậm chí còn ghi nhận cả những nước đi chưa bao giờ xảy ra! Ngược lại, các phương pháp hàm giá trị cho phép đánh giá các trạng thái riêng lẻ. Cuối cùng, các phương pháp hàm giá trị & tiến hóa đều tìm kiếm không gian của các chính sách, nhưng việc học 1 hàm giá trị sẽ tận dụng thông tin có sẵn trong quá trình chơi.

This simple example illustrates some of key features of RL methods. 1st, there is emphasis on learning while interacting with an environment in this case with an opponent player. 2nd, there is a clear goal, & correct behavior requires planning or foresight that takes into account delayed effects of one's choices. E.g., simple RL player would learn to set up multi-move traps for a shortsighted opponent. It is a striking feature of RL solution that it can achieve effects of planning & lookahead without using a model of opponent & without conducting an explicit search over possible sequences of future states & actions.

– Ví dụ đơn giản này minh họa 1 số đặc điểm chính của phương pháp RL. Thứ nhất, phương pháp này nhấn mạnh vào việc học hỏi trong khi tương tác với môi trường, trong trường hợp này là với đối thủ. Thứ hai, có 1 mục tiêu rõ ràng, & hành vi đúng đắn đòi hỏi sự lập kế hoạch hoặc tầm nhìn xa, có tính đến các tác động chậm trễ của lựa chọn. Ví dụ, 1 người chơi RL đơn giản sẽ học cách đặt bẫy nhiều nước đi cho 1 đối thủ thiển cận. Một đặc điểm nổi bật của giải pháp RL là nó có thể đạt được hiệu quả của việc lập kế hoạch & nhìn trước mà không cần sử dụng mô hình của đối thủ & không cần thực hiện tìm kiếm rõ ràng trên các chuỗi trạng thái & hành động có thể xảy ra trong tương lai.

While this example illustrates some of key features of RL, so simple: it might give impression that RL is more limited than it really is. Although tic-tac-toe is a 2-person game, RL also applies in case in which there is no external adversary, i.e., in case of a "game against nature". RL also is not restricted to problems in which behavior breaks down into separate episodes, like separate games of tic-tac-toe, with reward only at end of each episode. It is just as applicable when behavior continues indefinitely & when rewards of various magnitudes can be received at any time. RL is also applicable to problems that do not even break down into discrete time steps like plays of tic-tac-toe. General principles apply to continuous-time problems as well, although theory gets more complicated & omit it from this introductory treatment.

– Trong khi ví dụ này minh họa 1 số tính năng chính của RL, rất đơn giản: nó có thể tạo ấn tượng rằng RL bị hạn chế hơn thực tế. Mặc dù tic-tac-toe là trò chơi 2 người, RL cũng áp dụng trong trường hợp không có đối thủ bên ngoài, tức là trong trường hợp "trò chơi chống lại tự nhiên". RL cũng không bị giới hạn ở các vấn đề trong đó hành vi bị chia thành các tập riêng biệt, như các ván tic-tac-toe riêng biệt, với phần thưởng chỉ ở cuối mỗi tập. Nó cũng áp dụng khi hành vi tiếp tục vô thời hạn & khi phần thưởng có nhiều mức độ khác nhau có thể được nhận bất cứ lúc nào. RL cũng áp dụng cho các vấn đề thậm chí không bị chia thành các bước thời gian rời rạc như các ván tic-tac-toe. Các nguyên tắc chung cũng áp dụng cho các vấn đề thời gian liên tục, mặc dù lý thuyết trở nên phức tạp hơn & bỏ qua nó khỏi cách xử lý giới thiệu này.

Tic-tact-toe has a relatively small, finite state set, whereas RL can be used when state set is very large, or even infinite. E.g., Gerry Tesauro (1992, 1995) combined algorithm described above with an ANN to learn to play backgammon, which has $\approx 10^{20}$ states. With this many states, impossible ever to experience more than a small fraction of them. Tesauro's program learned to play far better than any previous program & eventually better than world's best human players (see Sect. 16.1). ANN provides program with ability to generalize from its experience, so that in new states it selects moves based on information saved from similar states faced in past, as determined by network. How well a RL system can work in problems with such large state sets is intimately tied to how appropriately it can generalize from past experiences. It is in this role: we have greatest need for supervised learning methods within RL. ANNs & DL (Sect. 9.7) are not the only, or necessary the best, way to do this.

– Tic-tact-toe có tập trạng thái hữu hạn tương đối nhỏ, trong khi RL có thể được sử dụng khi tập trạng thái rất lớn hoặc thậm chí vô hạn. Ví dụ, Gerry Tesauro (1992, 1995) đã kết hợp thuật toán được mô tả ở trên với ANN để học chơi cờ thỏ cáo, có $\approx 10^{20}$ trạng thái. Với nhiều trạng thái như vậy, không bao giờ có thể trải nghiệm nhiều hơn 1 phần nhỏ trong số chúng. Chương trình của Tesauro đã học cách chơi tốt hơn nhiều so với bất kỳ chương trình nào trước đó & cuối cùng là tốt hơn những người chơi giỏi nhất thế giới (xem Mục 16.1). ANN cung cấp cho chương trình khả năng khái quát hóa từ kinh nghiệm của chính nó, do đó, trong các trạng thái mới, nó sẽ chọn các nước đi dựa trên thông tin được lưu từ các trạng thái tương tự đã gặp phải trong quá khứ, do mạng xác định. Mức độ hoạt động tốt của 1 hệ thống RL trong các bài toán có tập trạng thái lớn như vậy gắn liền mật thiết với mức độ phù hợp mà nó có thể khái quát hóa từ các kinh nghiệm trong quá khứ. Trong vai trò này: chúng ta có nhu cầu lớn nhất về các phương pháp học có giám sát trong RL. ANN & DL (Mục 9.7) không phải là cách duy nhất hoặc tốt nhất để thực hiện điều này.

In this tic-tac-toe example, learning started with no prior knowledge beyond rules of game, but RL by no means entails a tabula rasa view of learning & intelligence. On contrary, prior information can be incorporated into RL in a variety of ways that can be critical for efficient learning (e.g., see Sects. 9.5, 17.4, & 13.1). Also have access to true state in tic-tac-toe example, whereas RL can also be applied when part of state is hidden, or when different states appear to learner to be same.

– Trong ví dụ cờ caro này, việc học bắt đầu mà không có kiến thức nền nào ngoài luật chơi, nhưng RL không hề hàm ý 1 cái nhìn phiến diện về học tập & trí tuệ. Ngược lại, thông tin nền tảng có thể được tích hợp vào RL theo nhiều cách khác nhau, rất quan trọng cho việc học tập hiệu quả (ví dụ: xem Mục 9.5, 17.4, & 13.1). Cũng có thể truy cập vào trạng thái thực trong ví dụ cờ caro, trong khi RL cũng có thể được áp dụng khi 1 phần trạng thái bị ẩn, hoặc khi các trạng thái khác nhau xuất hiện giống nhau đối với người học.

Finally, tic-tac-toe player was able to look ahead & know states that would result from each of its possible moves. To do this, it had to have a model of game that allowed it to foresee how its environment would change in response to moves that it might never make. Many problems are like this, but in others even a short-term model of effects of actions is lacking. RL can be applied in either case. A model is not required, but models can easily be used if they are available or can be learned (Chap. 8).

– Cuối cùng, người chơi cờ caro đã có thể nhìn trước & biết được các trạng thái sẽ xảy ra từ mỗi nước đi khả dĩ của mình. Để làm được điều này, người chơi phải có 1 mô hình trò chơi cho phép dự đoán môi trường xung quanh sẽ thay đổi như thế nào để phản ứng với những nước đi mà có thể họ sẽ không bao giờ thực hiện. Nhiều bài toán tương tự như vậy, nhưng ở 1 số bài toán khác, ngay cả 1 mô hình ngắn hạn về tác động của các hành động cũng còn thiếu. RL có thể được áp dụng trong cả hai trường hợp. Không bắt buộc phải có mô hình, nhưng có thể dễ dàng sử dụng các mô hình nếu có sẵn hoặc có thể học được (Chương 8).

On other hand, there are RL methods that do not need any kind of environment model at all. Model-free systems cannot even think about how their environments will change in response to a single action. Tic-tac-toe player is model-free in this sense w.r.t. its opponent: it has no model of its opponent of any kind. Because models have to be reasonably accurate to be useful, model-free methods can have advantages over more complex methods when real bottleneck in solving a problem is difficulty of constructing a sufficiently accurate environment model. Model-free methods are also important building blocks for model-based methods. In this book, devote several chaps to model-free methods before discuss how they can be used as components of more complex model-based methods.

– Mặt khác, có những phương pháp RL không cần bất kỳ loại mô hình môi trường nào cả. Các hệ thống không mô hình thậm chí không thể nghĩ về cách môi trường của chúng sẽ thay đổi để đáp ứng với 1 hành động duy nhất. Người chơi Tic-tac-toe là không mô hình theo nghĩa này đối với đối thủ của nó: nó không có mô hình của đối thủ dưới bất kỳ hình thức nào. Vì các mô hình phải có độ chính xác hợp lý để hữu ích, các phương pháp không mô hình có thể có lợi thế hơn các phương pháp phức tạp hơn khi nút thắt thực sự trong việc giải quyết vấn đề là khó khăn trong việc xây dựng 1 mô hình môi trường đủ chính xác. Các phương pháp không mô hình cũng là những khối xây dựng quan trọng cho các phương pháp dựa trên mô hình. Trong cuốn sách này, hãy dành 1 số chương về các phương pháp không mô hình trước khi thảo luận về cách chúng có thể được sử dụng như các thành phần của các phương pháp dựa trên mô hình phức tạp hơn.

RL can be used at both high & low levels in a system. Although tic-tac-toe player learned only about basic moves of game, nothing prevents RL from working at higher levels where each of "actions" may itself be application of a possibly elaborate problem-solving method. In hierarchical learning systems, RL can work simultaneously on several levels.

– Học tăng cường có thể được sử dụng ở cả cấp độ cao & thấp trong 1 hệ thống. Mặc dù người chơi cờ caro chỉ học được những nước đi cơ bản của trò chơi, nhưng không có gì ngăn cản học tăng cường hoạt động ở các cấp độ cao hơn, nơi mỗi "hành động" có thể tự nó là ứng dụng của 1 phương pháp giải quyết vấn đề phức tạp. Trong các hệ thống học phân cấp, học tăng cường có thể hoạt động đồng thời ở nhiều cấp độ.

Problem 1 (Self-Play). *Suppose, instead of playing against a random opponent, RL algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?*

– Giả sử, thay vì chơi với 1 đối thủ ngẫu nhiên, thuật toán RL được mô tả ở trên chơi với chính nó, cả hai bên đều học. Bạn nghĩ điều gì sẽ xảy ra trong trường hợp này? Liệu nó có học được 1 chính sách khác để lựa chọn nước đi không?

Problem 2 (Symmetries). *Many tic-tac-toe positions appear different but are really same because of symmetries. How might we amend learning process described above to take advantage of this? In what ways would this change improve learning process? Now think again. Suppose opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have same value?*

– Nhiều vị trí cờ caro trông có vẻ khác nhau nhưng thực chất lại giống nhau do tính đối xứng. Làm thế nào chúng ta có thể sửa đổi quy trình học tập được mô tả ở trên để tận dụng điều này? Sự thay đổi này sẽ cải thiện quy trình học tập theo những cách nào? Giờ hãy nghĩ lại. Giả sử đối thủ không tận dụng tính đối xứng. Trong trường hợp đó, chúng ta có nên làm vậy không? Vậy thì, liệu các vị trí tương đương đối xứng nhất thiết phải có cùng giá trị không?

Problem 3 (Greedy Play). *Suppose RL player was greedy, i.e., it always played the move that brought it to position that it rated best. Might it learn to play better, or worse, than a nongreedy player? What problems might occur?*

– Giả sử người chơi RL tham lam, tức là luôn đi nước đi đưa mình đến vị trí được đánh giá cao nhất. Liệu nó có thể học cách chơi tốt hơn, hay tệ hơn, 1 người chơi không tham lam? Những vấn đề nào có thể xảy ra?

Problem 4 (Learning from Exploration). *Suppose learning updates occurred after* all *moves, including exploratory moves. If step-size parameter is appropriately reduced over time (but not tendency to explore), then state values would converge to a different set of probabilities. What (conceptually) are 2 sets of probabilities computed when we do, & when we do not, learn from exploratory moves? Assuming: we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?*

– Giả sử các cập nhật học tập xảy ra sau tất cả *bước di chuyển, bao gồm cả các bước di chuyển thăm dò. Nếu tham số bước di chuyển được giảm phù hợp theo thời gian (nhưng không phải xu hướng khám phá), thì các giá trị trạng thái sẽ hội tụ về 1 tập hợp xác suất khác. Vậy (về mặt khái niệm), 2 tập hợp xác suất nào được tính toán khi chúng ta thực hiện, & khi chúng ta không thực hiện, học từ các bước di chuyển thăm dò? Giả sử: chúng ta tiếp tục thực hiện các bước di chuyển thăm dò, tập hợp xác suất nào có thể học tốt hơn? Tập hợp nào sẽ mang lại nhiều chiến thắng hơn?*

Problem 5 (Other Improvements). *Can you think of other ways to improve RL player? Can you think of any better way to solve tic-tac-toe problem as posed?*

– Bạn có thể nghĩ ra cách nào khác để cải thiện trình chơi RL không? Bạn có thể nghĩ ra cách nào tốt hơn để giải bài toán tic-tac-toe như đã nêu không?

- 1.6. Summary. RL is a computational approach to understanding & automating goal-directed learning & decision making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with

its environment, without requiring exemplary supervision or complete models of environment. In our opinion, RL is 1st field to seriously address computational issues that arise when learning from interaction with an environment in order to achieve long-term goals.

– RL là 1 phương pháp tính toán để hiểu & tự động hóa việc học tập hướng mục tiêu & ra quyết định. Phương pháp này khác biệt với các phương pháp tính toán khác ở chỗ nhấn mạnh vào việc học tập của tác nhân thông qua tương tác trực tiếp với môi trường, mà không cần giám sát mẫu mực hay mô hình môi trường hoàn chỉnh. Theo chúng tôi, RL là lĩnh vực đầu tiên nghiêm túc giải quyết các vấn đề tính toán phát sinh khi học tập từ tương tác với môi trường nhằm đạt được các mục tiêu dài hạn.

RL uses formal framework of Markov decision processes to define interaction between a learning agent & its environment in terms of states, actions, & rewards. This framework is intended to be a simple way of representing essential features of AI problem. These features include a sense of cause & effect, a sense of uncertainty & nondeterminism, & existence of explicit goals.

– RL sử dụng khuôn khổ chính thức của các quy trình quyết định Markov để xác định tương tác giữa tác nhân học tập & môi trường của nó dưới dạng trạng thái, hành động, & phần thưởng. Khuôn khổ này được thiết kế để biểu diễn các đặc điểm thiết yếu của bài toán AI 1 cách đơn giản. Các đặc điểm này bao gồm ý thức về nguyên nhân & kết quả, ý thức về sự không chắc chắn & tính không xác định, & sự tồn tại của các mục tiêu rõ ràng.

Concepts of value & value function are key to most of RL methods considered in this book. Take position that value functions are important for efficient search in space of policies. Use of value functions distinguishes RL methods from evolutionary methods that search directly in policy space guided by evaluations of entire policies.

– Khái niệm giá trị & hàm giá trị là chìa khóa cho hầu hết các phương pháp RL được đề cập trong cuốn sách này. Chúng tôi cho rằng hàm giá trị rất quan trọng để tìm kiếm hiệu quả trong không gian chính sách. Việc sử dụng hàm giá trị phân biệt các phương pháp RL với các phương pháp tiến hóa tìm kiếm trực tiếp trong không gian chính sách được hướng dẫn bởi việc đánh giá toàn bộ chính sách.

- 1.7. Early History of RL.

PART I: TABULAR SOLUTION METHODS. In this part of book, describe almost all core ideas of RL algorithms in their simplest forms: that in which state & action spaces are small enough for approximate value functions to be represented as arrays, or *tables*. In this case, methods can often find exact solutions, i.e., they can often find exactly optimal value function & optimal policy. This contrasts with approximate methods described in next part of book, which only find approximate solutions, but which in return can be applied effectively to much larger problems.

– Trong phần này của sách, chúng tôi sẽ mô tả hầu hết các ý tưởng cốt lõi của thuật toán RL ở dạng đơn giản nhất: trong đó không gian trạng thái & hành động đủ nhỏ để các hàm giá trị xấp xỉ được biểu diễn dưới dạng mảng hoặc *table*. Trong trường hợp này, các phương pháp thường có thể tìm ra giải pháp chính xác, tức là chúng thường có thể tìm ra chính xác hàm giá trị tối ưu & chính sách tối ưu. Điều này trái ngược với các phương pháp xấp xỉ được mô tả trong phần tiếp theo của sách, vốn chỉ tìm ra giải pháp xấp xỉ, nhưng đổi lại có thể được áp dụng hiệu quả cho các bài toán lớn hơn nhiều.

1st chap of this part of book describes solution methods for special case of RL problem in which there is only a single state, called *bandit problems*. 2nd chap describes general problem formulation that we treat throughout rest of book – finite Markov decision processes – & its main ideas including Bellman equations & value functions.

– Chương đầu tiên của phần này mô tả các phương pháp giải cho trường hợp đặc biệt của bài toán RL trong đó chỉ có 1 trạng thái duy nhất, được gọi là *bandit problems*. Chương thứ 2 mô tả cách xây dựng bài toán chung mà chúng tôi xử lý trong suốt phần còn lại của cuốn sách – quy trình quyết định Markov hữu hạn – & các ý tưởng chính bao gồm phương trình Bellman & hàm giá trị.

Next 3 chaps describe 3 fundamental classes of methods for solving finite Markov decision problems: dynamic programming, Monte Carlo methods, & temporal-difference learning. Each class of methods has its strengths & weaknesses. Dynamic programming methods are well developed mathematically, but require a complete & accurate model of environment. Monte Carlo methods don't require a model & are conceptually simple, but are not well suited for step-by-step incremental computation. Finally, temporal-difference methods require no model & are fully incremental, but are more complex to analyze. Methods also differ in several ways w.r.t. their efficiency & speed of convergence.

– 3 chương tiếp theo sẽ mô tả 3 lớp phương pháp cơ bản để giải quyết các bài toán quyết định Markov hữu hạn: quy hoạch động, phương pháp Monte Carlo, & học sai phân thời gian. Mỗi lớp phương pháp đều có những ưu điểm & nhược điểm riêng. Các phương pháp quy hoạch động được phát triển tốt về mặt toán học, nhưng đòi hỏi 1 mô hình môi trường hoàn chỉnh & chính xác. Các phương pháp Monte Carlo không yêu cầu mô hình & đơn giản về mặt khái niệm, nhưng không phù hợp cho tính toán gia tăng từng bước. Cuối cùng, các phương pháp sai phân thời gian không yêu cầu mô hình & hoàn toàn gia tăng, nhưng phức tạp hơn để phân tích. Các phương pháp cũng khác nhau về hiệu quả & tốc độ hội tụ.

Remaining 2 chaps describe how these 3 classes of methods can be combined to obtain best features of each of them. In 1 chap, describe how strengths of Monte Carlo methods can be combined with strengths of temporal-difference methods via multi-step bootstrapping methods. In final chap of this part of book, show how temporal-difference learning methods can be combined with model learning & planning methods (e.g. dynamic programming) for a complete & unified solution to tabular RL problem.

– Hai chương còn lại mô tả cách kết hợp 3 lớp phương pháp này để thu được các đặc điểm tốt nhất của từng lớp. Trong chương 1, mô tả cách kết hợp điểm mạnh của phương pháp Monte Carlo với điểm mạnh của phương pháp sai phân thời gian thông

qua phương pháp khởi động nhiều bước. Trong chương cuối cùng của phần này, trình bày cách kết hợp các phương pháp học sai phân thời gian với phương pháp học mô hình & phương pháp lập kế hoạch (ví dụ: quy hoạch động) để có được giải pháp & thống nhất hoàn chỉnh cho bài toán RL dạng bảng.

- 2. Multi-armed Bandits. Most important feature distinguishing RL from other types of learning: RL uses training information that *evaluates* actions taken rather than *instructs* by giving correct actions. This is what creates need for active exploration, for an explicit search for good behavior. Purely evaluative feedback indicates how good action taken was, but not whether it was best or worst action possible. Purely instructive feedback, on other hand, indicates correct action to take, independently of action actually taken. This kind of feedback is basis of supervised learning, which includes large parts of pattern classification, ANNs, & system identification. In their pure forms, these 2 kinds of feedback are quite distinct: evaluative feedback depends entirely on action taken, whereas instructive feedback is independent of action taken.

  –Máy đánh bạc nhiều tay. Đặc điểm quan trọng nhất phân biệt RL với các loại hình học tập khác: RL sử dụng thông tin đào tạo để *đánh giá* các hành động được thực hiện thay vì *hướng dẫn* bằng cách đưa ra các hành động đúng. Đây là điều tạo ra nhu cầu khám phá tích cực, để tìm kiếm rõ ràng hành vi tốt. Phản hồi đánh giá thuần túy chỉ ra hành động tốt được thực hiện như thế nào, nhưng không phải là hành động tốt nhất hay tệ nhất có thể. Mặt khác, phản hồi hướng dẫn thuần túy chỉ ra hành động đúng cần thực hiện, độc lập với hành động thực sự được thực hiện. Loại phản hồi này là cơ sở của học tập có giám sát, bao gồm phần lớn phân loại mẫu, ANN & nhận dạng hệ thống. Ở dạng thuần túy, 2 loại phản hồi này khá khác biệt: phản hồi đánh giá hoàn toàn phụ thuộc vào hành động được thực hiện, trong khi phản hồi hướng dẫn độc lập với hành động được thực hiện.

  In this chap, study evaluative aspect of RL in a simplified setting, one that does not involve learning to act in more than 1 situation. This *nonassociative* setting is the one in which most prior work involving evaluative feedback has been done, & it avoids much of complexity of full RL problem. Studying this case enables us to see most clearly how evaluative feedback differs from, & yet can be combined with, instructive feedback.

  – Trong chương này, chúng ta sẽ nghiên cứu khía cạnh đánh giá của RL trong 1 bối cảnh đơn giản, không liên quan đến việc học cách hành động trong nhiều hơn 1 tình huống. Bối cảnh *nonassociative* này là bối cảnh mà hầu hết các nghiên cứu trước đây liên quan đến phản hồi đánh giá đã được thực hiện, & nó tránh được phần lớn sự phức tạp của bài toán RL đầy đủ. Nghiên cứu trường hợp này cho phép chúng ta thấy rõ nhất sự khác biệt giữa phản hồi đánh giá & & nhưng có thể kết hợp với phản hồi hướng dẫn.

  Particular nonassociative, evaluative feedback problem that we explore is a simple version of k-armed bandit problem. Use this problem to introduce a number of basic learning methods which we extend in later chaps to apply to full RL problem. At end of this chap, take a step closer to full RL problem by discussing what happens when bandit problem becomes associative, i.e., when best action depends on situation.

  – Bài toán phản hồi đánh giá, phi liên kết cụ thể mà chúng ta đang tìm hiểu là 1 phiên bản đơn giản của bài toán máy đánh bạc k. Sử dụng bài toán này để giới thiệu 1 số phương pháp học cơ bản mà chúng tôi sẽ mở rộng trong các chương sau để áp dụng cho bài toán thực tế ảo đầy đủ. Cuối chương này, hãy tiến gần hơn 1 bước đến bài toán thực tế ảo đầy đủ bằng cách thảo luận về điều gì xảy ra khi bài toán máy đánh bạc trở thành bài toán kết hợp, tức là khi hành động tốt nhất phụ thuộc vào tình huống.

  - 2.1. A k-armed Bandit Problem. Consider following learning problem. You are faced repeatedly with a choice among k different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on action you selected. Your objective: maximize expected total reward over some time period, e.g., > 1000 action selections, or *time steps*.

    – Bài toán Cướp có vũ trang k. Hãy xem xét bài toán học sau. Bạn liên tục phải đối mặt với việc lựa chọn giữa k phương án, hay hành động, khác nhau. Sau mỗi lựa chọn, bạn nhận được 1 phần thưởng số được chọn từ phân phối xác suất dừng phụ thuộc vào hành động bạn đã chọn. Mục tiêu của bạn: tối đa hóa tổng phần thưởng kỳ vọng trong 1 khoảng thời gian nhất định, ví dụ: > 1000 lựa chọn hành động, hay *bước thời gian*.

    This is original form of *k-armed bandit problem*, so named by analogy to a slot machine, or "1-armed bandit", except: it has k levers instead of 1. Each action selection is like a play of 1 of slot machine's levers, & rewards are payoffs for hitting jackpot. Through repeated action selections you are to maximize your winnings by concentrating your actions on best levers. Another analogy is that of a doctor choosing between experimental treatments for a series of seriously ill patients. Each action is selection of a treatment, & each reward is survival or well-being of patient. Today term "bandit problem" is sometimes used for a generalization of problem described above, but in this book we use it to refer just to this simple case.

    – Đây là dạng gốc của *k-armed bandit problem*, được đặt tên tương tự như máy đánh bạc, hay "máy đánh bạc 1 tay", ngoại trừ: nó có k đòn bẩy thay vì 1. Mỗi lựa chọn hành động giống như 1 lần chơi 1 trong những đòn bẩy của máy đánh bạc, & phần thưởng là khoản tiền thưởng khi trúng giải độc đắc. Thông qua các lựa chọn hành động lặp lại, bạn phải tối đa hóa tiền thắng của mình bằng cách tập trung hành động của mình vào những đòn bẩy tốt nhất. Một phép tương tự khác là 1 bác sĩ lựa chọn giữa các phương pháp điều trị thử nghiệm cho 1 loạt bệnh nhân bị bệnh nặng. Mỗi hành động là lựa chọn 1 phương pháp điều trị, & mỗi phần thưởng là sự sống sót hoặc sức khỏe của bệnh nhân. Ngày nay, thuật ngữ "bài toán máy đánh bạc" đôi khi được sử dụng để khái quát hóa vấn đề được mô tả ở trên, nhưng trong cuốn sách này, chúng tôi sử dụng nó để chỉ trường hợp đơn giản này.

    In our k-armed bandit problem, each of k actions has an expected or mean reward given that that action is selected; call this *value* of that action. Denote action selected on time step t as A_t, & corresponding reward as R_t. Value then of an arbitrary

action a, denoted $q_*(a)$, is expected reward given that a is selected:

$$q_*(a) \coloneqq \mathbb{E}[R_t|A_t = a].$$

If you knew value of each action, then it would be trivial to solve k-armed bandit problem: you would always select action with highest value. Assume: you do not know action values with certainty, although you may have estimates. Denote estimated value of action a at time step t as $Q_t(a)$. Would like $Q_t(a)$ to be close to $q_*(a)$.

– Trong bài toán máy đánh bạc k của chúng ta, mỗi hành động trong số k hành động đều có phần thưởng kỳ vọng hoặc phần thưởng trung bình nếu hành động đó được chọn; hãy gọi *giá trị* của hành động đó. Ký hiệu hành động được chọn tại bước thời gian t là A_t, & phần thưởng tương ứng là R_t. Khi đó, giá trị của 1 hành động a bất kỳ, ký hiệu là $q_*(a)$, là phần thưởng kỳ vọng nếu a được chọn:

$$q_*(a) \coloneqq \mathbb{E}[R_t|A_t = a].$$

Nếu bạn biết giá trị của từng hành động, thì việc giải bài toán máy đánh bạc k sẽ rất đơn giản: bạn sẽ luôn chọn hành động có giá trị cao nhất. Giả sử: bạn không biết chắc chắn giá trị của hành động, mặc dù bạn có thể có ước tính. Ký hiệu giá trị ước tính của hành động a tại bước thời gian t là $Q_t(a)$. Muốn $Q_t(a)$ gần với $q_*(a)$.

If maintain estimates of action values, then at any time step there is at least 1 action whose estimated value is greatest. Call these *greedy* actions. When select 1 of these actions, say: you are *exploring* your current knowledge of values of actions. If instead you select 1 of nongreedy actions, then we say you are *exploring*, because this enables you to improve your estimate of nongreedy action's value. Exploitation is right thing to do to maximize expected reward on 1 step, but exploration may produce greater total reward in long run. E.g., suppose a greedy action's value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty. Uncertainty is s.t. at least 1 of these other actions probably is actually better than greedy actions, but you don't know which one. If you have many times steps ahead on which to make action selections, then it may be better to explore nongreedy actions & discover which of them are better than greedy action. Reward is lower in short run, during exploration, but higher in long run because after you have discovered better actions, you can exploit them many times. Because not possible both to explore & exploit with any single action selection, one often refers to "conflict" between exploration & exploitation.

– Nếu duy trì ước tính giá trị hành động, thì tại bất kỳ bước thời gian nào cũng có ít nhất 1 hành động có giá trị ước tính lớn nhất. Gọi những hành động này là *tham lam*. Khi chọn 1 trong những hành động này, hãy nói: bạn đang *khám phá* kiến thức hiện tại của mình về giá trị của các hành động. Thay vào đó, nếu bạn chọn 1 trong những hành động không tham lam, thì chúng ta nói bạn đang *khám phá*, vì điều này cho phép bạn cải thiện ước tính của mình về giá trị của hành động không tham lam. Khai thác là điều đúng đắn cần làm để tối đa hóa phần thưởng mong đợi ở 1 bước, nhưng khám phá có thể tạo ra tổng phần thưởng lớn hơn về lâu dài. Ví dụ: giả sử giá trị của 1 hành động tham lam được biết chắc chắn, trong khi 1 số hành động khác được ước tính là gần tốt nhưng có sự không chắc chắn đáng kể. Sự không chắc chắn là s.t. ít nhất 1 trong những hành động khác này có thể thực sự tốt hơn hành động tham lam, nhưng bạn không biết hành động nào. Nếu bạn có nhiều bước gấp nhiều lần để đưa ra lựa chọn hành động, thì tốt hơn là nên khám phá các hành động không tham lam & khám phá xem hành động nào trong số chúng tốt hơn hành động tham lam. Phần thưởng thấp hơn trong ngắn hạn, trong quá trình khám phá, nhưng cao hơn về lâu dài vì sau khi bạn khám phá ra những hành động tốt hơn, bạn có thể khai thác chúng nhiều lần. Vì không thể vừa khám phá & khai thác chỉ với 1 lựa chọn hành động duy nhất, nên người ta thường nói đến "xung đột" giữa khám phá & khai thác.

In any specific case, whether better to explore or exploit depends in a complex way on precise values of estimates, uncertainties, & number of remaining steps. There are many sophisticated methods for balancing exploration & exploitation for particular mathematical formulations of k-armed bandit & related problems. However, most of these methods make strong assumptions about stationary & prior knowledge that are either violated or impossible to verify in most applications & in full RL problem that we consider in subsequent chaps. Guarantees of optimality or bounded loss for these methods are of little comfort when assumptions of their theory do not apply.

– Trong bất kỳ trường hợp cụ thể nào, việc khám phá hay khai thác tốt hơn phụ thuộc rất nhiều vào các giá trị ước lượng chính xác, độ bất định, & số bước còn lại. Có nhiều phương pháp tinh vi để cân bằng giữa khám phá & khai thác cho các công thức toán học cụ thể của máy đánh bạc k-armed bandit & các bài toán liên quan. Tuy nhiên, hầu hết các phương pháp này đều đưa ra những giả định mạnh mẽ về kiến thức dừng & kiến thức đã biết, những giả định này bị vi phạm hoặc không thể xác minh trong hầu hết các ứng dụng & trong bài toán RL đầy đủ mà chúng ta sẽ xem xét trong các chương tiếp theo. Việc đảm bảo tính tối ưu hoặc tổn thất giới hạn cho các phương pháp này không mấy an toàn khi các giả định của lý thuyết của chúng không được áp dụng.

In this book, do not worry about balancing exploration & exploitation in a sophisticated way; worry only about balancing them at all. In this chap, present several simple balancing methods for k-armed bandit problem & show they work much better than methods that always exploit. Need to balance exploration & exploitation is a distinctive challenge that arises in RL; simplicity of our version of k-armed bandit problem enables us to show this in a particularly clear form.

– Trong cuốn sách này, đừng lo lắng về việc cân bằng giữa khám phá & khai thác 1 cách phức tạp; chỉ cần quan tâm đến việc cân bằng chúng. Trong chương này, chúng tôi trình bày 1 số phương pháp cân bằng đơn giản cho bài toán k-armed bandit & cho thấy chúng hiệu quả hơn nhiều so với các phương pháp luôn khai thác. Nhu cầu cân bằng giữa khám phá & khai thác là 1 thách thức đặc biệt phát sinh trong thực tế; sự đơn giản của phiên bản bài toán k-armed bandit của chúng tôi cho phép chúng tôi thể hiện điều này 1 cách đặc biệt rõ ràng.

- 2.2. Action-value Methods. begin by looking more closely at methods for estimating values of actions & for using estimates to make action selection decisions, which we collectively call *action-value methods*. Recall: true value of an action is mean

reward when that action is selected. 1 natural way to estimate this is by averaging rewards actually received: (2.1)

$$Q_t(a) \coloneqq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \cdot \mathbf{1}_{A_i=a}},$$

where $\mathbf{1}_{\text{predicate}}$ denotes random variable that is 1 if *predicate* is true & 0 if it is not. If denominator is 0, then instead define $Q_t(a)$ as some default value, e.g. 0. As denominator goes to ∞, by law of large numbers, $Q_t(a)$ converges to $q_*(a)$. Call this *sample-average* method for estimating action values because each estimate is an average of sample of relevant rewards. Of course this is just 1 way to estimate action values, & not necessarily best one. Nevertheless, for now, say with this simple estimation method & turn to question of how estimates might be used to select actions.

– Phương pháp Hành động-Giá trị. bắt đầu bằng cách xem xét kỹ hơn các phương pháp ước tính giá trị của hành động & sử dụng ước tính để đưa ra quyết định lựa chọn hành động, mà chúng tôi gọi chung là *phương pháp hành động-giá trị*. Nhắc lại: giá trị thực của 1 hành động là phần thưởng trung bình khi hành động đó được chọn. Một cách tự nhiên để ước tính điều này là lấy trung bình phần thưởng thực tế nhận được: (2.1)

$$Q_t(a) \coloneqq \frac{\text{tổng phần thưởng khi } a \text{ được lấy trước } t}{\text{số lần } a \text{ được lấy trước } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \cdot \mathbf{1}_{A_i=a}},$$

trong đó $\mathbf{1}_{\text{predicate}}$ biểu thị biến ngẫu nhiên có giá trị 1 nếu *predicate* là đúng & 0 nếu sai. Nếu mẫu số là 0, thì thay vào đó hãy định nghĩa $Q_t(a)$ là 1 giá trị mặc định, ví dụ: 0. Khi mẫu số tiến đến ∞, theo quy luật số lớn, $Q_t(a)$ hội tụ về $q_*(a)$. Gọi đây là phương pháp *sample-average* để ước tính giá trị hành động vì mỗi ước tính là trung bình của 1 mẫu phần thưởng có liên quan. Tất nhiên, đây chỉ là 1 cách để ước tính giá trị hành động, & không nhất thiết là cách tốt nhất. Tuy nhiên, hiện tại, hãy thử sử dụng phương pháp ước tính đơn giản này & chuyển sang câu hỏi về cách sử dụng ước tính để lựa chọn hành động.

Simplest action selection rule: select 1 of actions with highest estimated value, i.e., 1 of greedy actions as defined in prev sect. If there is > 1 greedy action, then a selection is made among them in some arbitrary way, perhaps randomly. Write this *greedy* action selection method as (2.2)

$$A_t \coloneqq \arg\max_a Q_t(a),$$

where $\arg\max_a$ denotes action a for which expression that follows is maximized (with ties broken arbitrarily). Greedy action selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative: behave greedily most of time, but every once in a while, say with small probability ε, instead select randomly from among all actions with equal probability, independently of action-value estimates. Call methods using this near-greedy action selection rule *ε-greedy* methods. An advantage of these methods: in limit as number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all $Q_t(a)$ converge to their respective $q_*(a)$. This of course implies: probability of selecting optimal action converges to $> 1 - \varepsilon$, i.e., to near certainty. These are just asymptotic guarantees, however, & say little about practical effectiveness of methods.

– Quy tắc lựa chọn hành động đơn giản nhất: chọn 1 trong các hành động có giá trị ước tính cao nhất, tức là 1 trong các hành động tham lam như được định nghĩa trong phần trước. Nếu có > 1 hành động tham lam, thì việc lựa chọn được thực hiện giữa chúng theo 1 cách tùy ý, có thể là ngẫu nhiên. Viết phương pháp lựa chọn hành động *tham lam* này dưới dạng (2.2)

$$A_t \coloneqq \arg\max_a Q_t(a),$$

trong đó $\arg\max_a$ biểu thị hành động a mà biểu thức theo sau được tối đa hóa (với các ràng buộc bị phá vỡ 1 cách tùy ý). Lựa chọn hành động tham lam luôn khai thác kiến thức hiện có để tối đa hóa phần thưởng tức thời; nó không mất thời gian để lấy mẫu các hành động có vẻ kém hơn để xem liệu chúng có thực sự tốt hơn hay không. Một giải pháp thay thế đơn giản: hành động tham lam hầu hết thời gian, nhưng thỉnh thoảng, với xác suất ε nhỏ, hãy chọn ngẫu nhiên từ tất cả các hành động có xác suất bằng nhau, độc lập với ước tính giá trị hành động. Gọi các phương thức sử dụng quy tắc lựa chọn hành động gần tham lam này *ε-tham lam*. Một ưu điểm của các phương thức này: khi số bước tăng lên, mỗi hành động sẽ được lấy mẫu vô hạn lần, do đó đảm bảo rằng tất cả $Q_t(a)$ đều hội tụ về $q_*(a)$ tương ứng của chúng. Điều này tất nhiên ngụ ý: xác suất lựa chọn hành động tối ưu hội tụ về $> 1 - \varepsilon$, tức là gần như chắc chắn. Tuy nhiên, đây chỉ là những đảm bảo tiệm cận, & không nói lên nhiều về hiệu quả thực tế của các phương thức.

Problem 6. *In ε-greedy action selection, for case of 2 actions & $\varepsilon = 0.5$, what is probability that greedy action is selected?*
– Trong lựa chọn hành động tham lam ε, đối với trường hợp 2 hành động & $\varepsilon = 0,5$, xác suất hành động tham lam được chọn là bao nhiêu?

- 2.3. 10-armed Testbed. To roughly assess relative effectiveness of greedy & ε-greedy action-value methods, compared them numerically on a suite of test problems. This was a set of 2000 randomly generated k-armed bandit problems with $k = 10$. For each bandit problem, e.g. one shown in Fig. 2.1: An example bandit problem from 10-armed testbed. True value $q_*(a)$ of esach of 10 actions was selected according to a normal distribution with mean 0 & unit variance, & then actual rewards were selected according to a mean $q_*(a)$, unit-variance normal distribution, as suggested by these gray distributions., action values, $q_*(a)$, $a \in [10]$, were selected according to a normal (Gaussian) distribution with mean 0 & variance 1. Then, when a learning method applied to that problem selected action A_t at time step t, actual reward R_t was selected from a normal distribution with mean $q_*(A_t)$ & variance 1. These distributions are shown in gray in Fig. 2.1. Call this suite of test tasks *10-armed*

testbed. For any learning method, can measure its performance & behavior as it improves with experience > 1000 time steps when applied to 1 of bandit problems. This makes up 1 *run*. Repeating this for 2000 independent runs, each with a different bandit problem, obtained measures of learning algorithm's average behavior.

– 10-armed Testbed. Để đánh giá sơ bộ hiệu quả tương đối của các phương pháp hành động-giá trị tham lam & ε-greedy, chúng tôi đã so sánh chúng bằng số trên 1 bộ bài toán thử nghiệm. Đây là 1 bộ gồm 2000 bài toán bandit k được tạo ngẫu nhiên với $k = 10$. Đối với mỗi bài toán bandit, ví dụ như bài toán được hiển thị trong Hình 2.1: Một ví dụ về bài toán bandit từ testbed 10-armed. Giá trị thực $q_*(a)$ của mỗi 10 hành động được chọn theo phân phối chuẩn với trung bình 0 & phương sai đơn vị, & sau đó phần thưởng thực tế được chọn theo trung bình $q_*(a)$, phân phối chuẩn phương sai đơn vị, như được gợi ý bởi các phân phối xám này., các giá trị hành động, $q_*(a)$, $a \in [10]$, được chọn theo phân phối chuẩn (Gaussian) với trung bình 0 & phương sai 1. Sau đó, khi 1 phương pháp học được áp dụng cho vấn đề đó đã chọn hành động A_t tại bước thời gian t, phần thưởng thực tế R_t được chọn từ phân phối chuẩn với trung bình $q_*(A_t)$ & phương sai 1. Các phân phối này được hiển thị bằng màu xám trong Hình 2.1. Gọi bộ tác vụ kiểm tra này là *10-armed testbed*. Đối với bất kỳ phương pháp học nào, có thể đo lường hiệu suất & hành vi của nó khi nó cải thiện theo kinh nghiệm > 1000 bước thời gian khi được áp dụng cho 1 trong số các bài toán bandit. Điều này tạo nên 1 *run*. Lặp lại điều này trong 2000 lần chạy độc lập, mỗi lần chạy với 1 bài toán cướp khác nhau, thu được các phép đo về hành vi trung bình của thuật toán học.

Fig. 2.2: Average performance of ε-greedy action-value methods on 10-armed testbed. These data are averages > 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates. compares a greedy method with 2 ε-greedy methods ($\varepsilon = 0.01, \varepsilon = 0.1$), as described above, on 10-armed testbed. All methods formed their action-value estimates using sample-average technique (with an initial estimate of 0). Upper graph shows increase in expected reward with experience. Greedy method improved slightly faster than order methods at very beginning, but then leveled off at a lower level. It achieved a reward-per-step of only about 1, compared with best possible of abut 1.54 on this testbed. Greedy method performed significantly worse in long run because it often got struck performing suboptimal actions. Lower graph shows: greedy method found optimal action in only $\approx \frac{1}{3}$ of tasks. In other $\frac{2}{3}$, its initial samples of optimal action were disappointing, & it never returned to it. ε-greedy methods eventually performed better because they continued to explore & to improve their chances of recognizing optimal action. $\varepsilon = 0.1$ method explored more, & usually found optimal action earlier, but it never selected that action $> 91\%$ of time. $\varepsilon = 0.01$ method improved more slowly, but eventually would perform better than $\varepsilon = 0.1$ method on both performance measures shown in figure. Also possible to reduce ε over time to try to get best of both high & low values.

– Hình 2.2: Hiệu suất trung bình của các phương pháp giá trị hành động ε-tham lam trên nền tảng thử nghiệm 10 cánh tay. Những dữ liệu này là trung bình > 2000 lần chạy với các vấn đề bandit khác nhau. Tất cả các phương pháp đều sử dụng trung bình mẫu làm ước tính giá trị hành động của chúng. so sánh 1 phương pháp tham lam với 2 phương pháp tham lam ε ($\varepsilon = 0,01, \varepsilon = 0,1$), như mô tả ở trên, trên nền tảng thử nghiệm 10 cánh tay. Tất cả các phương pháp đều hình thành ước tính giá trị hành động của chúng bằng kỹ thuật trung bình mẫu (với ước tính ban đầu là 0). Đồ thị trên cho thấy phần thưởng kỳ vọng tăng lên theo kinh nghiệm. Phương pháp tham lam cải thiện nhanh hơn 1 chút so với các phương pháp thứ tự ở giai đoạn đầu, nhưng sau đó ổn định ở mức thấp hơn. Nó đạt được phần thưởng cho mỗi bước chỉ khoảng 1, so với mức tốt nhất có thể là khoảng 1,54 trên nền tảng thử nghiệm này. Phương pháp tham lam hoạt động kém hơn đáng kể trong thời gian dài vì nó thường bị tấn công khi thực hiện các hành động không tối ưu. Biểu đồ bên dưới cho thấy: phương pháp tham lam chỉ tìm thấy hành động tối ưu trong $\approx \frac{1}{3}$ nhiệm vụ. Trong $\frac{2}{3}$ khác, các mẫu hành động tối ưu ban đầu của nó không được như mong đợi, & nó không bao giờ quay lại với nó. Các phương pháp ε-tham lam cuối cùng đã hoạt động tốt hơn vì chúng tiếp tục khám phá & để cải thiện cơ hội nhận ra hành động tối ưu. Phương pháp $\varepsilon = 0.1$ đã khám phá nhiều hơn, & thường tìm thấy hành động tối ưu sớm hơn, nhưng nó không bao giờ chọn hành động đó $> 91\%$ thời gian. Phương pháp $\varepsilon = 0.01$ cải thiện chậm hơn, nhưng cuối cùng sẽ hoạt động tốt hơn phương pháp $\varepsilon = 0.1$ trên cả hai thước đo hiệu suất được hiển thị trong hình. Cũng có thể giảm ε theo thời gian để cố gắng đạt được kết quả tốt nhất ở cả giá trị cao & thấp.

Advantage of ε-greedy over greedy methods depends on task. E.g., suppose reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find optimal action, & ε-greedy methods should fare even better relative to greedy method. On other hand, if reward variances were 0, then greedy method would know true value of each action after trying it once. In this case, greedy method might actually perform best because it would soon find optimal action & then never explore. But even in deterministic case there is a a large advantage to exploring if we weaken some of other assumptions. E.g., suppose bandit task were nonstationary, i.e., true values of actions changed over time. In this case exploration is needed even in deterministic case to make sure 1 of nongreedy actions has not changed to become better than greedy one. As will see in next few chaps, nonstationarity is case most commonly encountered in RL. Even if underlying task is stationary & deterministic, learner faces a set of banditlike decision tasks each of which changes over time as learning proceeds & agent's decision-making policy changes. RL requires a balance between exploration & exploitation.

– Ưu điểm của phương pháp ε-tham lam so với phương pháp ε-tham lam phụ thuộc vào nhiệm vụ. Ví dụ, giả sử phương sai phần thưởng lớn hơn, chẳng hạn 10 thay vì 1. Với phần thưởng nhiễu hơn, cần nhiều lần khám phá hơn để tìm ra hành động tối ưu, & phương pháp ε-tham lam thậm chí còn tốt hơn phương pháp tham lam. Mặt khác, nếu phương sai phần thưởng bằng 0, thì phương pháp tham lam sẽ biết giá trị thực của mỗi hành động sau khi thử 1 lần. Trong trường hợp này, phương pháp tham lam thực sự có thể hoạt động tốt nhất vì nó sẽ sớm tìm ra hành động tối ưu & sau đó không bao giờ khám phá. Nhưng ngay cả trong trường hợp xác định, việc khám phá vẫn có 1 lợi thế lớn nếu chúng ta làm suy yếu 1 số giả định khác. Ví dụ, giả sử nhiệm vụ cướp không dừng, tức là giá trị thực của các hành động thay đổi theo thời gian. Trong trường hợp này, việc khám phá là cần thiết ngay cả trong trường hợp xác định để đảm bảo rằng 1 trong các hành động không tham lam không thay đổi để trở nên tốt hơn hành động tham lam. Như sẽ thấy trong vài chương tiếp theo, tính không dừng là

trường hợp thường gặp nhất trong RL. Ngay cả khi nhiệm vụ cơ bản là tĩnh & xác định, người học vẫn phải đối mặt với 1 loạt các nhiệm vụ quyết định giống như trò chơi cướp bóc, mỗi nhiệm vụ thay đổi theo thời gian khi quá trình học diễn ra & chính sách ra quyết định của tác nhân thay đổi. RL đòi hỏi sự cân bằng giữa khám phá & khai thác.

Problem 7. *In comparison shown in Fig. 2.2, which method will perform best in long run in terms of cumulative reward & probability of selecting best action? How much better will it be? Express your answer quantitatively.*

– So sánh với Hình 2.2, phương pháp nào sẽ hiệu quả hơn về lâu dài xét về phần thưởng tích lũy & xác suất lựa chọn hành động tốt nhất? Phương pháp đó sẽ tốt hơn bao nhiêu? Hãy thể hiện câu trả lời của bạn bằng định lượng.

- 2.4. Incremental Implementation. Action-value methods discussed so far all estimate action values as sample averages of observed rewards. Now turn to question of how these averages can be computed in a computationally efficient manner, in particular, with constant memory & constant per-time-step computation.

  – Triển khai Gia tăng. Các phương pháp giá trị hành động đã thảo luận cho đến nay đều ước tính giá trị hành động dưới dạng trung bình mẫu của phần thưởng quan sát được. Bây giờ hãy chuyển sang câu hỏi làm thế nào để tính toán các giá trị trung bình này 1 cách hiệu quả về mặt tính toán, đặc biệt là với bộ nhớ không đổi & tính toán không đổi theo từng bước thời gian.

  To simplify notation, concentrate on a single action. Let R_i now denote reward received after ith selection of *this action*, & let Q_n denote estimate of its action value after it has been selected $n-1$ times, which we can now write simply as

$$Q_n \coloneqq \frac{\sum_{i=1}^{n-1} R_i}{n-1}.$$

  Obvious implementation would be maintain a record of all rewards & then perform this computation whenever estimated value was needed. However, if this is done, then memory & computational requirements would grow over time as more rewards are seen. Each additional reward would require additional memory to store it & additional computation to compute sum in numerator.

  – Để đơn giản hóa ký hiệu, hãy tập trung vào 1 hành động duy nhất. Giả sử R_i biểu thị phần thưởng nhận được sau lần chọn thứ i của *this action*, & giả sử Q_n biểu thị ước tính giá trị hành động của nó sau khi được chọn $n-1$ lần, mà giờ đây chúng ta có thể viết đơn giản là

$$Q_n \coloneqq \frac{\sum_{i=1}^{n-1} R_i}{n-1}.$$

  Cách triển khai rõ ràng sẽ là duy trì 1 bản ghi của tất cả phần thưởng & sau đó thực hiện phép tính này bất cứ khi nào cần giá trị ước tính. Tuy nhiên, nếu thực hiện điều này, thì bộ nhớ & yêu cầu tính toán sẽ tăng theo thời gian khi có thêm phần thưởng. Mỗi phần thưởng bổ sung sẽ yêu cầu thêm bộ nhớ để lưu trữ & thêm phép tính để tính tổng trong tử số.

  As you might suspect, this is not really necessary. Easy to devise incremental formulas for updating averages with small, constant computation required to process each new reward. Given Q_n & nth reward R_n, new average of all n rewards can be computed by (2.3)

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \cdots = Q_n + \frac{R_n - Q_n}{n},$$

  which holds even for $n=1$, obtaining $Q_2 = R_1$ for arbitrary Q_1. This implementation requires memory only for Q_n & n, & only small computation (2.3) for each new reward.

  – Như bạn có thể nghi ngờ, điều này thực sự không cần thiết. Dễ dàng thiết kế các công thức gia tăng để cập nhật giá trị trung bình với các phép tính nhỏ, không đổi cần thiết để xử lý mỗi phần thưởng mới. Với Q_n & n phần thưởng thứ R_n, giá trị trung bình mới của tất cả n phần thưởng có thể được tính theo (2.3)

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \cdots = Q_n + \frac{R_n - Q_n}{n},$$

  điều này đúng ngay cả với $n=1$, thu được $Q_2 = R_1$ cho Q_1 bất kỳ. Việc triển khai này chỉ yêu cầu bộ nhớ cho Q_n & n, & chỉ cần phép tính nhỏ (2.3) cho mỗi phần thưởng mới.

  This update rule (2.3) is of a form that occurs frequently throughout this book. General form is (2.4)

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}].$$

  Expression [Target − OldEstimate] is an *error* in estimate. It is reduced by taking a step toward "Target". Target is presumed to indicate a desirable direction in which to move, thought it may be noisy. In case above, e.g., target is nth reward.

  – Quy tắc cập nhật (2.3) này có dạng thường xuyên xuất hiện trong suốt cuốn sách này. Dạng tổng quát là (2.4)

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}].$$

  Biểu thức [Target − OldEstimate] là 1 *error* trong ước tính. Nó được rút gọn bằng cách thực hiện 1 bước tiến về phía "Target". Target được cho là chỉ ra 1 hướng mong muốn để di chuyển, mặc dù nó có thể bị nhiễu. Trong trường hợp trên, ví dụ, target là phần thưởng thứ n.

Note: step-size parameter StepSize used in incremental method (2.3) changes from time step to time step. In processing nth reward for action a, method uses step-size parameter $\frac{1}{n}$. In this book, denote step-size parameter by α or, more generally, by $\alpha_t(a)$.

– Lưu ý: tham số kích thước bước StepSize được sử dụng trong phương pháp gia tăng (2.3) thay đổi theo từng bước thời gian. Khi xử lý phần thưởng thứ n cho hành động a, phương pháp sử dụng tham số kích thước bước $\frac{1}{n}$. Trong sách này, hãy ký hiệu tham số kích thước bước là α hoặc, tổng quát hơn, là $\alpha_t(a)$.

Pseucode for a complete bandit algorithm using incrementally computed sample averages & ε-greedy action selection is shown in box below. Function bandit(a) is assumed to take an action & return a corresponding reward.

– Mã giả cho 1 thuật toán cướp hoàn chỉnh sử dụng trung bình mẫu được tính toán gia tăng & lựa chọn hành động tham lam ε được hiển thị trong hộp bên dưới. Hàm bandit(a) được giả định thực hiện 1 hành động & trả về phần thưởng tương ứng.

- 2.5. Tracking a Nonstationary Problem. Averaging methods discussed so far are appropriate for stationary bandit problems, i.e., for bandit problems in which reward probabilities do not change over time. As noted earlier, often encounter RL problems that are effectively nonstationary. In such cases it makes sense to give more weight to recent rewards than to long-past rewards. 1 of most popular ways of doing this: use a constant step-size parameter. E.g., incremental update rule (2.3) for updating an average Q_n of $n-1$ past rewards is modified to be

$$Q_{n+1} \coloneqq Q_n + \alpha[R_n - Q_n],$$

where step-size parameter $\alpha \in (0, 1]$ is constant. This results in Q_{n+1} being a weighted average of past rewards & initial estimate Q_1: (2.6)

$$Q_{n+1} = \cdots = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.$$

Call this a weighted average because sum of weights is $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$. Note: weight $\alpha(1-\alpha)^{n-i}$, given to reward R_i depends on how many rewards ago, $n-i$, it was observed. Quantity $1-\alpha < 1$, & thus weight given to R_i decreases as number of intervening rewards increases. In fact, weight decays exponentially according to exponent on $1-\alpha$. (If $1-\alpha = 0$, then all weight goes on very last reward R_n, because of convention that $O^0 = 1$.) Accordingly, this is sometimes called an *exponential recency-weighted average*.

– Theo dõi 1 Bài toán Không dừng. Các phương pháp tính trung bình đã thảo luận cho đến nay phù hợp với các bài toán bandit dừng, tức là các bài toán bandit trong đó xác suất phần thưởng không thay đổi theo thời gian. Như đã lưu ý trước đó, thường gặp phải các bài toán RL về cơ bản là không dừng. Trong những trường hợp như vậy, việc chú trọng hơn vào phần thưởng gần đây hơn là phần thưởng đã qua lâu là hợp lý. Một trong những cách phổ biến nhất để thực hiện việc này: sử dụng tham số kích thước bước không đổi. Ví dụ, quy tắc cập nhật gia tăng (2.3) để cập nhật Q_n trung bình của $n-1$ phần thưởng trong quá khứ được sửa đổi thành

$$Q_{n+1} \coloneqq Q_n + \alpha[R_n - Q_n],$$

trong đó tham số kích thước bước $\alpha \in (0, 1]$ là hằng số. Điều này dẫn đến Q_{n+1} là trung bình có trọng số của phần thưởng trong quá khứ & ước lượng ban đầu Q_1: (2.6)

$$Q_{n+1} = \cdots = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.$$

Gọi đây là trung bình có trọng số vì tổng trọng số là $(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$. Lưu ý: trọng số $\alpha(1-\alpha)^{n-i}$, được trao cho phần thưởng R_i phụ thuộc vào số phần thưởng trước đó, $n-i$, đã được quan sát. Số lượng $1-\alpha < 1$, & do đó trọng số được trao cho R_i giảm khi số phần thưởng xen kẽ tăng. Trên thực tế, trọng số giảm theo cấp số nhân theo số mũ của $1-\alpha$. (Nếu $1-\alpha = 0$, thì toàn bộ trọng số sẽ được chuyển sang phần thưởng cuối cùng R_n, do quy ước $O^0 = 1$.) Do đó, điều này đôi khi được gọi là *trung bình có trọng số gần đây theo cấp số nhân*.

Sometimes convenient to vary step-size parameter from step to step. Let $\alpha_n(a)$ denote step-size parameter used to process reward received after nth selection of action a. As have noted, choice $\alpha_n(a) = \frac{1}{n}$ results in sample-average method, which is guaranteed to converge to true action values by law of large numbers. But of course convergence is not guaranteed for all choices of sequence $\{\alpha_n(a)\}$. A well-known result in stochastic approximation theory gives us conditions required to assure convergence with probability 1: (2.7)

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty, \ \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$$

1st condition is required to guarantee: steps are large enough to eventually overcome any initial conditions or random fluctuations. 2nd condition guarantees that eventually steps become small enough to assure convergence.

– Đôi khi thuận tiện để thay đổi tham số kích thước bước từ bước này sang bước khác. Giả sử $\alpha_n(a)$ biểu thị tham số kích thước bước được sử dụng để xử lý phần thưởng nhận được sau lần lựa chọn hành động a thứ n. Như đã lưu ý, lựa chọn $\alpha_n(a) = \frac{1}{n}$ dẫn đến phương pháp trung bình mẫu, được đảm bảo hội tụ về giá trị hành động thực theo quy luật số lớn.

Nhưng tất nhiên, sự hội tụ không được đảm bảo cho mọi lựa chọn của chuỗi $\{\alpha_n(a)\}$. Một kết quả nổi tiếng trong lý thuyết xấp xỉ ngẫu nhiên cung cấp cho chúng ta các điều kiện cần thiết để đảm bảo sự hội tụ với xác suất 1: (2.7)

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty, \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$$

Điều kiện thứ nhất cần đảm bảo: các bước đủ lớn để cuối cùng vượt qua mọi điều kiện ban đầu hoặc biến động ngẫu nhiên. Điều kiện thứ hai đảm bảo rằng cuối cùng các bước sẽ đủ nhỏ để đảm bảo sự hội tụ.

Note both convergence conditions are met for sample-average case, $\alpha_n(a) = \frac{1}{n}$, but not for case of constant step-size parameter, $\alpha_n(a)$. In latter case, 2nd condition is not met, indicating: estimates never completely converge but2 continue to vary in response to most recently received rewards. As mentioned above, this is actually desirable in a nonstationary environment, & problems that are effectively nonstationary are most common in RL. In addition, sequences of step-size parameters that meet conditions (2.7) often converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate. Although sequences of step-size parameters that meet these convergence conditions are often used in theoretical work, they are seldom used in applications & empirical research.

– Lưu ý cả hai điều kiện hội tụ đều được đáp ứng đối với trường hợp trung bình mẫu, $\alpha_n(a) = \frac{1}{n}$, nhưng không đáp ứng đối với trường hợp tham số kích thước bước không đổi, $\alpha_n(a)$. Trong trường hợp sau, điều kiện thứ 2 không được đáp ứng, cho thấy: các ước tính không bao giờ hội tụ hoàn toàn mà2 tiếp tục thay đổi để đáp ứng với phần thưởng nhận được gần đây nhất. Như đã đề cập ở trên, điều này thực sự mong muốn trong môi trường không dừng, & các vấn đề thực sự không dừng là phổ biến nhất trong RL. Ngoài ra, các chuỗi tham số kích thước bước đáp ứng các điều kiện (2.7) thường hội tụ rất chậm hoặc cần điều chỉnh đáng kể để có được tốc độ hội tụ thỏa đáng. Mặc dù các chuỗi tham số kích thước bước đáp ứng các điều kiện hội tụ này thường được sử dụng trong công việc lý thuyết, nhưng chúng hiếm khi được sử dụng trong các ứng dụng & nghiên cứu thực nghiệm.

Problem 8. *If step-size parameters α_n are not constant, then estimate Q_n is a weighted average of previously received rewards with a weighting different from that given by (2.6). What is weighting on each prior reward for general case, analogous to (2.6), in terms of sequence of step-size parameters?*

– Nếu các tham số kích thước bước α_n không phải là hằng số, thì ước tính Q_n là giá trị trung bình có trọng số của các phần thưởng đã nhận trước đó với trọng số khác với trọng số được cho bởi (2.6). Trọng số của mỗi phần thưởng trước đó trong trường hợp tổng quát, tương tự như (2.6), là bao nhiêu về mặt trình tự các tham số kích thước bước?

Problem 9 (Programming)**.** *Design & conduct an experiment to demonstrate difficulties that sample-average methods have for nonstationary problems: Use a modified version of 10-armed testbed in which all $q_*(a)$ start out equal & then take independent random walks (say by adding a normally distributed increment with mean* 0 *& standard deviation* 0.01 *to all $q_*(a)$ on each step). Prepare plots like Fig. 2.2 for an action-value method using sample averages, incrementally computed, & another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\varepsilon = 0.1$ & longer runs, say of* 10000 *steps.*

– Thiết kế & tiến hành 1 thí nghiệm để chứng minh những khó khăn mà các phương pháp trung bình mẫu gặp phải đối với các vấn đề không dừng: Sử dụng phiên bản đã sửa đổi của nền tảng thử nghiệm 10 cánh tay trong đó tất cả $q_*(a)$ đều bắt đầu bằng nhau & sau đó thực hiện các bước đi ngẫu nhiên độc lập (ví dụ bằng cách thêm 1 gia số phân phối chuẩn với trung bình 0 *& độ lệch chuẩn* 0,01 *vào tất cả $q_*(a)$ ở mỗi bước). Chuẩn bị các biểu đồ như Hình 2.2 cho phương pháp giá trị hành động sử dụng trung bình mẫu, được tính toán gia tăng, & 1 phương pháp giá trị hành động khác sử dụng tham số kích thước bước không đổi, $\alpha = 0,1$. Sử dụng $\varepsilon = 0,1$ & các lần chạy dài hơn, ví dụ* 10000 *bước.*

- 2.6. Optimistic Initial Values. All methods we have discussed so far are dependent to some extent on initial action-value estimates, $Q_1(a)$. In language of statistics, these methods are *biased* by their initial estimates. For sample-average methods, bias disappears once all actions have been selected at least once, but for methods with constant α, bias is permanent, though decreasing over time as given by (2.6). In practice, this kind of bias is usually not a problem & can sometimes be very helpful. Downside: initial estimates become, in effect, a set of parameters that must be picked by user, if only to set them all to 0. Upside: they provide an easy way to supply some prior knowledge about what level of rewards can be expected.

  – Giá trị ban đầu lạc quan. Tất cả các phương pháp chúng ta đã thảo luận cho đến nay đều phụ thuộc ở 1 mức độ nào đó vào ước tính giá trị hành động ban đầu, $Q_1(a)$. Theo ngôn ngữ thống kê, các phương pháp này *bị sai lệch* bởi các ước tính ban đầu của chúng. Đối với các phương pháp trung bình mẫu, sai lệch biến mất khi tất cả các hành động đã được chọn ít nhất 1 lần, nhưng đối với các phương pháp có α không đổi, sai lệch là vĩnh viễn, mặc dù giảm dần theo thời gian như được cho bởi (2.6). Trong thực tế, loại sai lệch này thường không phải là vấn đề & đôi khi có thể rất hữu ích. Nhược điểm: các ước tính ban đầu, trên thực tế, trở thành 1 tập hợp các tham số phải được người dùng chọn, nếu chỉ để đặt tất cả chúng thành 0. Ưu điểm: chúng cung cấp 1 cách dễ dàng để cung cấp 1 số kiến thức trước đó về mức phần thưởng có thể mong đợi.

  Initial action values can also be used as a simple way to encourage exploration. Suppose: instead of setting initial action values to 0, as we did in 10-armed testbed, set them all to +5. Recall: $q_*(a)$ in this problem are selected from a normal distribution with mean 0 & variance 1. An initial estimate of +5 is thus wildly optimistic. But this optimism encourages action-value methods to explore. Whichever actions are initially selected, reward is less than starting estimates; learner switches to other actions, being "disappointed" with rewards it is receiving. Result: all actions are tried several times before value estimates converge. System does a fair amount of exploration even if greedy actions are selected all time.

  – Giá trị hành động ban đầu cũng có thể được sử dụng như 1 cách đơn giản để khuyến khích khám phá. Giả sử: thay vì đặt giá trị hành động ban đầu thành 0, như chúng ta đã làm trong thử nghiệm 10 cánh tay, hãy đặt tất cả chúng thành +5.

Nhớ lại: $q_*(a)$ trong bài toán này được chọn từ phân phối chuẩn với trung bình 0 & phương sai 1. Do đó, ước tính ban đầu là +5 là quá lạc quan. Nhưng sự lạc quan này khuyến khích các phương pháp giá trị hành động khám phá. Bất kỳ hành động nào được chọn ban đầu, phần thưởng sẽ thấp hơn ước tính ban đầu; người học chuyển sang các hành động khác, "thất vọng" với phần thưởng mà nó nhận được. Kết quả: tất cả các hành động được thử nhiều lần trước khi ước tính giá trị hội tụ. Hệ thống thực hiện 1 lượng khám phá kha khá ngay cả khi các hành động tham lam được chọn mọi lúc.

Fig. 2.3: Effect of optimistic initial action-value estimates on 10-armed testbed. Both methods used a constant step-size parameter $\alpha = 0.1$. shows performance on 10-armed bandit testbed of a greedy method using $Q_1(a) = +5$, $\forall a$. For comparison, also shown is an ε-greedy method with $Q_1(a) = 0$. Initially, optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time. Call this technique for encouraging exploration *optimistic initial values*. Regard it as a simple trick that can be quite effective on stationary problems, but it is far from being a generally useful approach to encouraging exploration. E.g., not well suited to nonstationary problem because its derive for exploration is inherently temporary. If task changes, creating a renewed need for exploration, this method cannot help. Indeed, any method that focuses on initial conditions in any special way is unlikely to help with general nonstationary case. Beginning of time occurs only once, & thus we should not focus on it too much. This criticism applies as well to sample-average methods, which also treat beginning of time as a special event, averaging all subsequent rewards with equal weights. Nevertheless, all of these methods are very simple, & 1 of them – or some simple combination of them – is often adequate in practice. In rest of this book, make frequent use of several of these simple exploration techniques.

– Hình 2.3: Ảnh hưởng của ước tính giá trị hành động ban đầu lạc quan trên nền tảng thử nghiệm 10 cánh tay. Cả hai phương pháp đều sử dụng tham số kích thước bước không đổi $\alpha = 0,1$. cho thấy hiệu suất trên nền tảng thử nghiệm 10 cánh tay của phương pháp tham lam sử dụng $Q_1(a) = +5$, $\forall a$. Để so sánh, cũng được hiển thị là phương pháp tham lam ε với $Q_1(a) = 0$. Ban đầu, phương pháp lạc quan hoạt động kém hơn vì nó khám phá nhiều hơn, nhưng cuối cùng nó hoạt động tốt hơn vì khả năng khám phá của nó giảm theo thời gian. Gọi kỹ thuật này là để khuyến khích khám phá *giá trị ban đầu lạc quan*. Hãy coi nó như 1 mẹo đơn giản có thể khá hiệu quả đối với các vấn đề dừng, nhưng nó còn lâu mới là 1 cách tiếp cận hữu ích để khuyến khích khám phá. Ví dụ: không phù hợp với vấn đề không dừng vì đạo hàm của nó để khám phá về cơ bản là tạm thời. Nếu nhiệm vụ thay đổi, tạo ra nhu cầu khám phá mới, phương pháp này không thể giúp ích. Thật vậy, bất kỳ phương pháp nào tập trung vào các điều kiện ban đầu theo bất kỳ cách đặc biệt nào đều khó có thể giúp ích cho trường hợp phi dừng tổng quát. Sự khởi đầu của thời gian chỉ xảy ra 1 lần, & do đó chúng ta không nên tập trung quá nhiều vào nó. Lời chỉ trích này cũng áp dụng cho các phương pháp trung bình mẫu, vốn cũng coi sự khởi đầu của thời gian là 1 sự kiện đặc biệt, tính trung bình tất cả các phần thưởng tiếp theo với trọng số bằng nhau. Tuy nhiên, tất cả các phương pháp này đều rất đơn giản, & 1 trong số chúng – hoặc 1 sự kết hợp đơn giản nào đó – thường là đủ trong thực tế. Trong phần còn lại của cuốn sách này, hãy thường xuyên sử dụng 1 số kỹ thuật khám phá đơn giản này.

Problem 10 (Mysterious Spikes)**.** *Results shown in Fig. 2.3 should be quite reliable because they are averages* > 2000 *individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations & spikes in early part of curve for optimistic method? I.e., what might make this method perform particularly better or worse, on average, on particular early steps?*

– Kết quả thể hiện trong Hình 2.3 khá đáng tin cậy vì chúng là các giá trị trung bình > 2000 *của các tác vụ máy đánh bạc 10 tay được chọn ngẫu nhiên. Vậy tại sao lại có dao động & gai ở phần đầu đường cong đối với phương pháp lạc quan? Nghĩa là, điều gì có thể khiến phương pháp này hoạt động tốt hơn hoặc kém hơn, xét về mặt trung bình, ở các bước đầu cụ thể?*

Problem 11 (Unbiased Constant-Step-Size Trick.)**.** *In most of this chap, have used sample averages to estimate action values because sample averages do not produce initial bias that constant step sizes do (see analysis leading to (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. Is it possible to avoid bias of constant step sizes while retaining their advantages on nonstationary problems? 1 way: use a step size of* $\beta_n \coloneqq \frac{\alpha}{\overline{o}_n}$*, to process* n*th reward for a particular action, where* $\alpha > 0$ *is a conventional constant step size, &* $\overline{o}_n$ *is a trace of one that starts at 0:*

$$\overline{o}_n \coloneqq \overline{o}_{n-1} + \alpha(1 - \overline{o}_{n-1}),\ \forall n \in \mathbb{N}^\star,\ \textit{with}\ \overline{o}_n \coloneqq 0.$$

Carry out an analysis like that in (2.6) to show: Q_n *is an exponential recency-weighted average* without initial bias.

– Trong hầu hết chương này, chúng tôi đã sử dụng trung bình mẫu để ước tính giá trị hành động vì trung bình mẫu không tạo ra độ lệch ban đầu như kích thước bước không đổi (xem phân tích dẫn đến (2.6)). Tuy nhiên, trung bình mẫu không phải là 1 giải pháp hoàn toàn thỏa đáng vì chúng có thể hoạt động kém trong các bài toán không dừng. Liệu có thể tránh được độ lệch của kích thước bước không đổi trong khi vẫn giữ được ưu điểm của chúng trong các bài toán không dừng không? Cách 1: sử dụng kích thước bước $\beta_n \coloneqq \frac{\alpha}{\overline{o}_n}$*, để xử lý phần thưởng thứ* n *cho 1 hành động cụ thể, trong đó* $\alpha > 0$ *là kích thước bước không đổi thông thường, &* $\overline{o}_n$ *là dấu vết của 1 bắt đầu từ 0:*

$$\overline{o}_n \coloneqq \overline{o}_{n-1} + \alpha(1 - \overline{o}_{n-1}),\ \forall n \in \mathbb{N}^\star,\ \textit{với}\ \overline{o}_n \coloneqq 0.$$

Thực hiện phân tích tương tự như trong (2.6) để chứng minh: Q_n *là trung bình trọng số mũ gần đây* không có độ lệch ban đầu.

◦ 2.7. Upper-Confidence-Bound Action Selection. Exploration is needed because there is always uncertainty about accuracy of action-value estimates. Greedy actions are those that look best at present, but some of other actions may actually be better. ε-greedy action selection forces non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain. It would be better to select among non-greedy actions according to their potential

for actually being optimal, taking into account both how close their estimates are to being maximal & uncertainties in those estimates. 1 effective way of doing this is to select actions according to (2.10)

$$A_t \coloneqq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

where $\ln t$ denotes natural logarithm of t, $N_t(a)$ denotes number of times that action a has been selected prior to time t (denominator in (2.1)), & number $c > 0$ controls degree of exploration. If $N_t(a) = 0$, then a is considered to be a maximizing action.

– Lựa chọn Hành động Giới hạn Độ tin cậy Trên. Việc khám phá là cần thiết vì luôn có sự không chắc chắn về độ chính xác của các ước tính giá trị hành động. Các hành động tham lam là những hành động có vẻ tốt nhất hiện tại, nhưng 1 số hành động khác thực sự có thể tốt hơn. Lựa chọn hành động tham lam ε buộc phải thử các hành động không tham lam, nhưng 1 cách bừa bãi, không ưu tiên những hành động gần tham lam hoặc đặc biệt không chắc chắn. Sẽ tốt hơn nếu lựa chọn giữa các hành động không tham lam theo tiềm năng tối ưu thực sự của chúng, đồng thời tính đến cả mức độ gần với ước tính tối đa của chúng & mức độ không chắc chắn trong các ước tính đó. Một cách hiệu quả để thực hiện điều này là chọn các hành động theo (2.10)

$$A_t \coloneqq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

trong đó $\ln t$ biểu thị logarit tự nhiên của t, $N_t(a)$ biểu thị số lần hành động a được chọn trước thời điểm t (mẫu số trong (2.1)), & số $c > 0$ kiểm soát mức độ khám phá. Nếu $N_t(a) = 0$, thì a được coi là 1 hành động tối đa hóa.

- 2.8. Gradient Bandit Algorithms.

- 3. Finite Markov Decision Processes.
- 4. Dynamic Programming.
- 5. Monte Carlo Methods.
- 6. Temporal-Difference Learning.
- 7. n-step Boostrapping.
- 8. Planning & Learning with Tabular Methods.

  PART II: APPROXIMATE SOLUTION METHODS.
- 9. On-policy Prediction with Approximation.
- 10. On-policy Control with Approximation.
- 11. Off-policy Methods with Approximation.
- 12. Eligibility Traces.
- 13. Policy Gradient Methods.

  PART III: LOOKING DEEPER.
- 14. Psychology.
- 15. Neuroscience.
- 16. Applications & Case Studies.
- 17. Frontiers.

2 Reinforcement Learning for Optimal Job Scheduling

2.1 Xinquan Wu, Xuefeng Yan, Mingqiang Wei, Donghai Guan. An Efficient Deep Reinforcement Learning Environment for Flexible Job-Shop Scheduling. Sep 10, 2025

- Abstract. Flexible Job-shop Scheduling Problem (FJSP) is a classical combinatorial optimization problem that has a wide-range of applications in real world. In order to generate fast & accurate scheduling solutions for FJSP, various deep reinforcement learning (DRL) scheduling methods have been developed. However, these methods are mainly focused on design of DRL scheduling Agent, overlooking modeling of DRL environment. This paper presents a simple chronological DRL environment for FJSP based on discrete event simulation & an end-to-end DRL scheduling model is proposed based on proximal policy optimization (PPO). Furthermore, a short novel state representation of FJSP is proposed based on 2 state variables in scheduling environment & a novel comprehensive reward function is designed based on scheduling area of machines. Experimental results

on public benchmark instances show: performance of simple priority dispatching rules (PDR) is improved in our scheduling environment & our DRL scheduling model obtains competing performance compared with OR-Tools, meta-heuristic, DRL & PDR scheduling methods.

– Bài toán Lập lịch Xưởng Linh hoạt (FJSP) là 1 bài toán tối ưu hóa tổ hợp cổ điển có phạm vi ứng dụng rộng rãi trong thực tế. Để tạo ra các giải pháp lập lịch nhanh & chính xác cho FJSP, nhiều phương pháp lập lịch học tăng cường sâu (DRL) đã được phát triển. Tuy nhiên, các phương pháp này chủ yếu tập trung vào thiết kế Tác nhân lập lịch DRL, bỏ qua việc mô hình hóa môi trường DRL. Bài báo này trình bày 1 môi trường DRL theo trình tự thời gian đơn giản cho FJSP dựa trên mô phỏng sự kiện rời rạc & 1 mô hình lập lịch DRL đầu cuối được đề xuất dựa trên tối ưu hóa chính sách gần đúng (PPO). Hơn nữa, 1 biểu diễn trạng thái mới ngắn gọn của FJSP được đề xuất dựa trên 2 biến trạng thái trong môi trường lập lịch & 1 hàm thưởng toàn diện mới được thiết kế dựa trên vùng lập lịch của máy. Kết quả thử nghiệm trên các trường hợp chuẩn công khai cho thấy: hiệu suất của các quy tắc phân bổ ưu tiên đơn giản (PDR) được cải thiện trong môi trường lập lịch của chúng tôi & mô hình lập lịch DRL của chúng tôi đạt được hiệu suất cạnh tranh so với các phương pháp lập lịch OR-Tools, meta-heuristic, DRL & PDR.

- 1. Introduction. Job shop scheduling problem (JSSP) is a classical NP-hard combinatorial optimization problem. It has been studied for decades & has been applied in a wide-range of areas including semiconductor, machine manufacturing, metallurgy, automobile manufacturing, supply chain & other fields [25]. Flexible Job-shop Scheduling Problem (FJSP) is a generalization of JSSP, which allows each operation to be processed on multiple candidate machines [6]. This makes it a more challenging problem with more complex topology & larger solution space.

  – Bài toán lập lịch xưởng gia công (JSSP) là 1 bài toán tối ưu tổ hợp NP-khó kinh điển. Nó đã được nghiên cứu trong nhiều thập kỷ & đã được ứng dụng trong nhiều lĩnh vực bao gồm bán dẫn, chế tạo máy, luyện kim, sản xuất ô tô, chuỗi cung ứng & các lĩnh vực khác [25]. Bài toán lập lịch xưởng gia công linh hoạt (FJSP) là 1 dạng tổng quát của JSSP, cho phép mỗi thao tác được xử lý trên nhiều máy ứng viên [6]. Điều này làm cho nó trở thành 1 bài toán khó hơn với cấu trúc phức tạp hơn & không gian nghiệm lớn hơn.

  Extensive researches have been widely explored to solve FJSP in recent years, including exact, heuristic, meta-heuristic, hyper-heuristic & RL methods. Exact approaches e.g. mixed integer linear programming (MILP) [20] may suffer from curse of dimension. So it is intractable to find exact scheduling solutions in a given time limit. Heuristic dispatching rules are widely used in real world manufacturing due to their simple & fast nature. Simple priority dispatching rules (PDR)[21], e.g. most work remaining (MWKR) possess advantages of high flexibility & easy implementation, but performance of PDR methods is not stable for different optimization objectives. Hyper-heuristic methods [29] e.g. genetic programming-based hyper-heuristic (GPHH), provide a way of automatically designing dispatching rules [28]. However, GP-evolved rules often has a large number of features in terminal set, making it difficult to identify promising search areas & determine optimal features.

  – Các nghiên cứu sâu rộng đã được khai thác rộng rãi để giải quyết FJSP trong những năm gần đây, bao gồm các phương pháp chính xác, heuristic, meta-heuristic, hyper-heuristic & RL. Các phương pháp chính xác, ví dụ như lập trình tuyến tính số nguyên hỗn hợp (MILP) [20] có thể bị lời nguyền về chiều. Do đó, việc tìm ra các giải pháp lập lịch chính xác trong 1 giới hạn thời gian nhất định là khó khăn. Các quy tắc điều phối heuristic được sử dụng rộng rãi trong sản xuất thực tế do bản chất đơn giản & nhanh chóng của chúng. Các quy tắc điều phối ưu tiên đơn giản (PDR) [21], ví dụ như hầu hết công việc còn lại (MWKR) có ưu điểm là tính linh hoạt cao & dễ triển khai, nhưng hiệu suất của các phương pháp PDR không ổn định đối với các mục tiêu tối ưu khác nhau. Các phương pháp hyper-heuristic [29], ví dụ như hyper-heuristic dựa trên lập trình di truyền (GPHH), cung cấp 1 cách để tự động thiết kế các quy tắc điều phối [28]. Tuy nhiên, các quy tắc phát triển GP thường có 1 số lượng lớn các tính năng trong tập đầu cuối, khiến việc xác định các khu vực tìm kiếm đầy hứa hẹn & xác định các tính năng tối ưu trở nên khó khăn.

  Different from PDR, meta-heuristic methods e.g. Tabu Search [12], Genetic Algorithm [19], Differential Evolution [7] & Particle Swarm Optimization [1], can produce higher solution quality than PDR due to introduction of iterative, randomized search strategies. Nevertheless, these methods also have shortcomings e.g. slow convergence speed, difficulty in obtaining global optimal solutions for large-scale problems & are difficult to apply to dynamic scenarios that need real-time decisions.

  – Khác với PDR, các phương pháp siêu heuristic như Tìm kiếm Tabu [12], Thuật toán Di truyền [19], Tiến hóa Vi phân [7] & Tối ưu hóa Bầy đàn Hạt [1] có thể tạo ra chất lượng giải pháp cao hơn PDR nhờ áp dụng các chiến lược tìm kiếm lặp lại, ngẫu nhiên. Tuy nhiên, các phương pháp này cũng có những hạn chế, ví dụ như tốc độ hội tụ chậm, khó đạt được giải pháp tối ưu toàn cục cho các bài toán quy mô lớn & khó áp dụng cho các tình huống động cần quyết định theo thời gian thực.

  RL is 1 of most important branches of ML & attracts interests of researchers from numerous fields especially for scheduling. Early RL scheduling methods represent scheduling policies using arrays or tables [2], which are only suitable for small-scale scheduling problems because of curse of dimension. However, scheduling tasks in real world usually have a higher dimensional state space & large action space, which limits applications of RL. Since deep neural networks have a strong fitting capability to process high-dimensional data, deep reinforcement learning (DRL) has been used to solve scheduling tasks with large or continuous state space & shows great potential to solve various scheduling problems. In DRL scheduling methods, scheduling policy is usually designed based on convolutional neural network (CNN) in [9], multi-layer perception (MLP) in [9, 10], RNN in [14], GNN in [15, 22, 26], attention networks in [23] & other deep neural networks in [16]. Scheduling agent is often trained by RL algorithms e.g. deep Q-learning (DQN), proximal policy optimization (PPO), Deep Deterministic Policy Gradient (DDPG). However, above DRL scheduling methods obtained low accurate solutions & some are even inferior to simple PDR. Even though recent GMAS model [13] whose policy is designed based on Graph Convolutional Network (GCN), obtained SOTA results on standard benchmark instances, it is a decentralized Multi-agent rRL (MARL) model. Besides, current DRL

scheduling methods focus mainly on design of scheduling agents, overlooking modeling of scheduling environment which is of vital important for improving performance of DRL scheduling methods.

– RL là 1 trong những nhánh quan trọng nhất của ML & thu hút sự quan tâm của các nhà nghiên cứu từ nhiều lĩnh vực, đặc biệt là đối với việc lập lịch. Các phương pháp lập lịch RL ban đầu biểu diễn các chính sách lập lịch sử dụng mảng hoặc bảng [2], chỉ phù hợp với các vấn đề lập lịch quy mô nhỏ do giới hạn chiều. Tuy nhiên, các tác vụ lập lịch trong thế giới thực thường có không gian trạng thái nhiều chiều & không gian hành động lớn, điều này hạn chế các ứng dụng của RL. Vì mạng nơ-ron sâu có khả năng khớp mạnh để xử lý dữ liệu nhiều chiều, nên học tăng cường sâu (DRL) đã được sử dụng để giải quyết các tác vụ lập lịch với không gian trạng thái lớn hoặc liên tục & cho thấy tiềm năng to lớn trong việc giải quyết nhiều vấn đề lập lịch khác nhau. Trong các phương pháp lập lịch DRL, chính sách lập lịch thường được thiết kế dựa trên mạng nơ-ron tích chập (CNN) trong [9], nhận thức đa lớp (MLP) trong [9, 10], RNN trong [14], GNN trong [15, 22, 26], mạng chú ý trong [23] & các mạng nơ-ron sâu khác trong [16]. Tác nhân lập lịch thường được đào tạo bằng các thuật toán RL, ví dụ: Học sâu Q (DQN), tối ưu hóa chính sách gần đúng (PPO), Gradient Chính sách Xác định Sâu (DDPG). Tuy nhiên, các phương pháp lập lịch DRL trên cho kết quả độ chính xác thấp & 1 số thậm chí còn kém hơn PDR đơn giản. Mặc dù mô hình GMAS gần đây [13], với chính sách được thiết kế dựa trên Mạng Tích chập Đồ thị (GCN), đã đạt được kết quả SOTA trên các trường hợp chuẩn mực, nhưng nó là 1 mô hình rRL Đa tác tử phi tập trung (MARL). Bên cạnh đó, các phương pháp lập lịch DRL hiện tại chủ yếu tập trung vào thiết kế các tác nhân lập lịch, bỏ qua việc mô hình hóa môi trường lập lịch, vốn rất quan trọng để cải thiện hiệu suất của các phương pháp lập lịch DRL.

There are mainly 4 modeling methods for FJSP: mathematical [20], Petri Nets (PN)-based [17], Disjunctive Graph (DG)-based [3] & simulation-based modeling [24]. Mathematical modeling methods usually formulates FJSP as a MILP & formulated problem is then be solved by optimization methods e.g. Genetic Algorithm. PN is a versatile modeling tool that can be used to model scheduling problems as they can model parallel activities, resource sharing & synchronization. Both modeling methods are not suitable for DRL scheduling methods. DG is another alternative to model FJSP which integrates machine status & operation dependencies, & provides critical structural information for scheduling decisions. However, DG fails to incorporate dynamically changing state information of FJSP, requiring extra handcrafted node features. Besides, allocation order provided by DG model is not strictly chronological. Simulation-based modeling method: develop algorithms to simulate laws of activities in real world. In current simulation environment of DRL scheduling methods, scheduling process is promoted by events in candidate queue, ignoring occurrence temporal order of these events.

– Có chủ yếu 4 phương pháp mô hình hóa cho FJSP: toán học [20], dựa trên Petri Nets (PN) [17], dựa trên Disjunctive Graph (DG) [3] & mô hình hóa dựa trên mô phỏng [24]. Các phương pháp mô hình hóa toán học thường xây dựng FJSP như 1 bài toán MILP & sau đó được giải quyết bằng các phương pháp tối ưu hóa, ví dụ như thuật toán di truyền. PN là 1 công cụ mô hình hóa đa năng có thể được sử dụng để mô hình hóa các vấn đề lập lịch vì chúng có thể mô hình hóa các hoạt động song song, chia sẻ tài nguyên & đồng bộ hóa. Cả hai phương pháp mô hình hóa đều không phù hợp với các phương pháp lập lịch DRL. DG là 1 giải pháp thay thế khác cho mô hình FJSP tích hợp trạng thái máy & phụ thuộc hoạt động, & cung cấp thông tin cấu trúc quan trọng cho các quyết định lập lịch. Tuy nhiên, DG không kết hợp thông tin trạng thái thay đổi động của FJSP, yêu cầu các tính năng nút thủ công bổ sung. Bên cạnh đó, thứ tự phân bổ do mô hình DG cung cấp không hoàn toàn theo thứ tự thời gian. Phương pháp mô hình hóa dựa trên mô phỏng: phát triển các thuật toán để mô phỏng các quy luật hoạt động trong thế giới thực. Trong môi trường mô phỏng hiện tại của các phương pháp lập lịch DRL, quá trình lập lịch được thúc đẩy bởi các sự kiện trong hàng đợi ứng viên, bỏ qua thứ tự thời gian xảy ra của các sự kiện này.

In this paper, a chronological DRL scheduling environment for FJSP is presented based on discrete event simulation algorithm where at each decision step, accurate state changes of scheduling process are recorded by state variables & a novel comprehensive reward function is designed based on scheduling area. Furthermore, an end to end DRL model for FJSP is proposed based on actor-critic PPO. Specially, a very short state representation is expressed by 2 state variables which are derived directly from necessary variables in environment simulation algorithm & action space is constructed using 6 PDRs from literature. Besides, in order to reduce computation time for RL agent, a simple scheduling policy is designed based on MLP with only 1 hidden layer & Softmax function.

– Bài báo này trình bày 1 môi trường lập lịch DRL theo thời gian cho FJSP dựa trên thuật toán mô phỏng sự kiện rời rạc, trong đó tại mỗi bước quyết định, các thay đổi trạng thái chính xác của quy trình lập lịch được ghi lại bởi các biến trạng thái & 1 hàm thưởng toàn diện mới được thiết kế dựa trên vùng lập lịch. Hơn nữa, 1 mô hình DRL đầu cuối cho FJSP được đề xuất dựa trên PPO tác nhân-phê bình. Cụ thể, 1 biểu diễn trạng thái rất ngắn được thể hiện bằng 2 biến trạng thái được suy ra trực tiếp từ các biến cần thiết trong thuật toán mô phỏng môi trường & không gian hành động được xây dựng bằng cách sử dụng 6 PDR từ tài liệu. Bên cạnh đó, để giảm thời gian tính toán cho tác nhân RL, 1 chính sách lập lịch đơn giản được thiết kế dựa trên MLP với chỉ 1 lớp ẩn & hàm Softmax.

Contributions of this work are listed as follows.

1. A basic simulation algorithm based on chronological discrete event is designed for FJSP, providing a general environment for DRL scheduling methods.
2. A simple DRL model for FJSP is proposed based on PPO where a very short state representation for FJSP is presented, avoiding handcrafted state features & massive experiments for feature selection & a novel comprehensive reward function is proposed based on scheduling area.
3. Experimental results show: performance of single PDR is improved in our environment, even better than some DRL scheduling methods & proposed DRL scheduling model obtains competing performance compared with OR-Tools, meta-heuristic, DRL & PDR scheduling methods.

– Những đóng góp của công trình này được liệt kê như sau.

1. 1 thuật toán mô phỏng cơ bản dựa trên sự kiện rời rạc theo trình tự thời gian được thiết kế cho FJSP, cung cấp 1 môi trường chung cho các phương pháp lập lịch DRL.
2. 1 mô hình DRL đơn giản cho FJSP được đề xuất dựa trên PPO, trong đó 1 biểu diễn trạng thái rất ngắn cho FJSP được trình bày, tránh các đặc trưng trạng thái thủ công & các thử nghiệm hàng loạt để lựa chọn đặc trưng & 1 hàm thưởng toàn diện mới được đề xuất dựa trên khu vực lập lịch.
3. Kết quả thử nghiệm cho thấy: hiệu suất của PDR đơn lẻ được cải thiện trong môi trường của chúng tôi, thậm chí còn tốt hơn 1 số phương pháp lập lịch DRL & mô hình lập lịch DRL được đề xuất đạt được hiệu suất cạnh tranh so với các phương pháp lập lịch OR-Tools, meta-heuristic, DRL & PDR.

Rest of paper is organized as follows. A chronological discrete event simulation-based scheduling environment for FJSP is introduced in Sect. II. In Sect. III, proposed DRL scheduling model for FJSP are constructed. Sect. IV demonstrates detailed experiments on standard benchmarks with different sizes & results are compared & discussed, while conclusions & future work reside in Sect. V.

– Phần còn lại của bài báo được tổ chức như sau. 1 môi trường lập lịch dựa trên mô phỏng sự kiện rời rạc theo thời gian cho FJSP được giới thiệu trong Phần II. Trong Phần III, mô hình lập lịch DRL đề xuất cho FJSP đã được xây dựng. Phần IV trình bày các thí nghiệm chi tiết trên các chuẩn mực chuẩn với các kích thước khác nhau & kết quả được so sánh & thảo luận, trong khi kết luận & các nghiên cứu trong tương lai nằm trong Phần V.

- 2. Simulation environment for FJSP. In this sect, 1st introduced basics of FJSPs. Then defined storage structure of FJSP & rules of state changing when scheduling. Finally, detailed simulation algorithm for FJSP is presented.

  – Môi trường mô phỏng cho FJSP. Trong phần này, trước tiên chúng tôi giới thiệu những kiến thức cơ bản về FJSP. Sau đó, chúng tôi định nghĩa cấu trúc lưu trữ của FJSP & các quy tắc thay đổi trạng thái khi lập lịch. Cuối cùng, thuật toán mô phỏng chi tiết cho FJSP được trình bày.

  - 2.1. Flexible Job-shop Scheduling Problems. FJSP differs from JSSP in that it allows an operation to be processed by any machine from a given set & different machines may have different processing times. Problem: assign each operation to eligible machine s.t. maximal completion time (makespan) of all jobs is minimized. It can be presented as $FJ||Cmax$ by 3-field notations [6].

    – FJSP khác với JSSP ở chỗ nó cho phép 1 thao tác được xử lý bởi bất kỳ máy nào trong 1 tập hợp nhất định & các máy khác nhau có thể có thời gian xử lý khác nhau. Vấn đề: gán mỗi thao tác cho máy đủ điều kiện, thời gian hoàn thành tối đa (makespan) của tất cả các công việc được giảm thiểu. Nó có thể được biểu diễn dưới dạng $FJ||Cmax$ theo ký hiệu 3 trường [6].

    In public benchmark of FJSP, environment information is passed through an instance file. As is depicted in Fig. 1: An example of flexible job-shop scheduling problem (MK1 [4]), 1st line in instance file consists of 3 integers representing number of jobs, number of machines & average number of machines per operation (optional), resp. Each of following lines describes information of a job where 1st integer is number of operations & followed integers depicts operation information. Operation information includes 2 integers: 1st represents index of a machine & 2nd is processing time of operation on this machine. Processing operation order in a job is advanced from left to right.

    – Trong chuẩn mực công khai của FJSP, thông tin môi trường được truyền qua 1 tệp thể hiện. Như được mô tả trong Hình 1: Ví dụ về bài toán lập lịch xưởng công việc linh hoạt (MK1 [4]), dòng đầu tiên trong tệp thể hiện bao gồm 3 số nguyên biểu diễn số lượng công việc, số lượng máy & số lượng máy trung bình trên mỗi thao tác (tùy chọn), tương ứng. Mỗi dòng sau mô tả thông tin của 1 công việc, trong đó số nguyên đầu tiên là số lượng thao tác & các số nguyên tiếp theo mô tả thông tin thao tác. Thông tin thao tác bao gồm 2 số nguyên: số 1 biểu diễn chỉ số của 1 máy & số 2 là thời gian xử lý thao tác trên máy này. Thứ tự thao tác trong 1 công việc được sắp xếp theo thứ tự từ trái sang phải.

  - 2.2. Data structure for FJSP. In this paper, based on benchmark instance file, propose a new & efficient storage structure for FJSP instances. Scheduling information is represented by a 2D table where job information is described in rows & column records processing stage information. Each element in this table includes 2 sets: machine set & remaining processing time set. Fig. 2: Storage structure of FJSP instance. demonstrates an example of a FJSP instance where storage structure is marked with a red box. It is efficient to retrieve any operation information using job index & processing stage index.

    – Cấu trúc dữ liệu cho FJSP. Trong bài báo này, dựa trên tệp phiên bản chuẩn, chúng tôi đề xuất 1 cấu trúc lưu trữ mới & hiệu quả cho các phiên bản FJSP. Thông tin lập lịch được biểu diễn bằng 1 bảng 2D, trong đó thông tin công việc được mô tả theo hàng & cột ghi lại thông tin giai đoạn xử lý. Mỗi phần tử trong bảng này bao gồm 2 tập hợp: tập máy & tập thời gian xử lý còn lại. Hình 2: Cấu trúc lưu trữ của phiên bản FJSP. minh họa 1 ví dụ về phiên bản FJSP, trong đó cấu trúc lưu trữ được đánh dấu bằng hộp đỏ. Việc truy xuất bất kỳ thông tin thao tác nào bằng chỉ mục công việc & chỉ mục giai đoạn xử lý đều hiệu quả.

  - 2.3. Rules of state updating. In order to accurately record state changes of each scheduling step, 4 rules of state updating are defined based on type of operations. Job operations are divided into 4 categories: operations on processing, waiting operations without predecessors, completed operations & operations whose predecessors have not been completed. 4 rules are listed as follows & Fig. 3: Use of state update rules on a FJSP instance at time 4 provides an example to demonstrate use of these rules.

– Quy tắc cập nhật trạng thái. Để ghi lại chính xác các thay đổi trạng thái của mỗi bước lập lịch, 4 quy tắc cập nhật trạng thái được xác định dựa trên loại thao tác. Các thao tác công việc được chia thành 4 loại: thao tác đang xử lý, thao tác chờ không có thao tác tiền nhiệm, thao tác đã hoàn thành & thao tác có thao tác tiền nhiệm chưa hoàn thành. 4 quy tắc được liệt kê như sau & Hình 3: Sử dụng quy tắc cập nhật trạng thái trên 1 thể hiện FJSP tại thời điểm 4 cung cấp 1 ví dụ để minh họa việc sử dụng các quy tắc này.

* Rule 1: For an operation on processing, machine set has only 1 element which is negative index of selected machine while remaining time is recorded in remaining processing time set & its value decreases as time advances until completion of this operation.
  – Quy tắc 1: Đối với 1 thao tác xử lý, tập hợp máy chỉ có 1 phần tử là chỉ số âm của máy được chọn trong khi thời gian còn lại được ghi vào tập hợp thời gian xử lý còn lại & giá trị của nó giảm dần theo thời gian cho đến khi hoàn thành thao tác này.
* Rule 2: For a waiting operation without predecessors whose needed machine is occupied, elements in machine set are all negative index of machines & elements in remaining processing time set stay unchanged. When needed machine is released, value of machine index is restored to be positive.
  – Quy tắc 2: Đối với 1 thao tác chờ không có tiền nhiệm mà máy cần thiết đang bị chiếm dụng, các phần tử trong tập máy đều có chỉ số máy âm & các phần tử trong tập thời gian xử lý còn lại không đổi. Khi máy cần thiết được giải phóng, giá trị chỉ số máy sẽ được khôi phục thành dương.
* Rule 3: For a completed operation, values in machine set & remaining processing time set equal to negative value of machine index & 0, resp.
  – Quy tắc 3: Đối với 1 hoạt động đã hoàn thành, các giá trị trong bộ máy & thời gian xử lý còn lại được đặt bằng giá trị âm của chỉ số máy & 0, tương ứng.
* Rule 4: For operations whose predecesors have not been proposed, values in machine set & remaining processing time set remain unchanged.
  – Quy tắc 4: Đối với các hoạt động mà hoạt động trước đó chưa được đề xuất, các giá trị trong tập hợp máy & thời gian xử lý còn lại vẫn không thay đổi.

This representation for FJSP instance can be recorded in a instance file & reloaded to new environment, i.e., it can be applied to nonzero or re-entrant scenarios.

– Biểu diễn này cho phiên bản FJSP có thể được ghi lại trong tệp phiên bản & tải lại vào môi trường mới, tức là có thể áp dụng cho các trường hợp khác không hoặc có thể nhập lại.

- 2.4. Simulation algorithm for FJSP. In this paper, proposed scheduling environment model is a simulation model based on chronological discrete events. There is a timer used to record current time & triggering time of events are recorded in a state variable. Once current time reaches triggering time of any events, this event will occur & corresponding event response program will be executed. Detailed process is illustrated in Algorithm 1: Simulation algorithm for FJSP.

  – Thuật toán mô phỏng cho FJSP. Trong bài báo này, mô hình môi trường lập lịch được đề xuất là 1 mô hình mô phỏng dựa trên các sự kiện rời rạc theo trình tự thời gian. Có 1 bộ đếm thời gian được sử dụng để ghi lại thời gian hiện tại & thời gian kích hoạt của các sự kiện được ghi lại trong 1 biến trạng thái. Khi thời gian hiện tại đạt đến thời gian kích hoạt của bất kỳ sự kiện nào, sự kiện này sẽ xảy ra & chương trình phản hồi sự kiện tương ứng sẽ được thực thi. Quy trình chi tiết được minh họa trong Thuật toán 1: Thuật toán mô phỏng cho FJSP.

  Proposed algorithm is mainly composed of 4 parts: selection of jobs & machines, state update of jobs & machines, time advance (3rd while statement) & machine release (4th while statement). In 1st part, decision action is divided into PDRs for job & machine selection & then specific job number & machine number are derived from PDRs. After allocating jobs & machines, their states are updated using state variables & table dictionaries. E.g., completion time of this job operation (or release time of machine) is recorded in `next_time_on_machine` & job-machine allocation information is recorded in `job_on_machine`. Once a job is allocated on a machine, state of this job is `assignable_job` changes to 0 & states of jobs which need that machine, are updated. If candidate machines of a job are all occupied by other jobs this job will become not assignable.

  – Thuật toán đề xuất chủ yếu bao gồm 4 phần: lựa chọn công việc & máy, cập nhật trạng thái của công việc & máy, tiến độ thời gian (câu lệnh while thứ 3) & giải phóng máy (câu lệnh while thứ 4). Trong phần 1, hành động quyết định được chia thành các PDR để lựa chọn công việc & máy & sau đó số công việc cụ thể & số máy được lấy từ PDR. Sau khi phân bổ công việc & máy, trạng thái của chúng được cập nhật bằng các biến trạng thái & từ điển bảng. Ví dụ: thời gian hoàn thành của hoạt động công việc này (hoặc thời gian giải phóng máy) được ghi lại trong `next_time_on_machine` & thông tin phân bổ công việc-máy được ghi lại trong `job_on_machine`. Sau khi 1 công việc được phân bổ trên 1 máy, trạng thái của công việc này là `assignable_job` thay đổi thành 0 & trạng thái của các công việc cần máy đó được cập nhật. Nếu tất cả các máy ứng viên của 1 công việc đều bị các công việc khác chiếm giữ thì công việc này sẽ không thể gán được.

  When there are no assignable jobs, time advance stage is coming. In this part, current time is 1st updated based on values in `next_time_on_machine` (if current time is smaller than any value in `next_time_on_machine`, take smallest value, otherwise take next smallest of them). Length of time step that needs to advance is then calculated based on `next_time_on_machine` & current time. Finally, next time of machines whose next time is slower than current time is advanced to current time. Machine release part releases machines whose release time reaches current time & updates status of jobs & machines. When current job operation is completed, occupied machine becomes idle & current operation of this job move to next operation. States of jobs & machines are updated in `assignable_job, job_on_machine`. If all operation of current job is completed, this job become not assignable & if machine needed for next operation is occupied, this job is still not assignable.

– Khi không có công việc nào có thể gán được, giai đoạn tiến độ thời gian sẽ diễn ra. Trong phần này, thời gian hiện tại được cập nhật lần đầu tiên dựa trên các giá trị trong `next_time_on_machine` (nếu thời gian hiện tại nhỏ hơn bất kỳ giá trị nào trong `next_time_on_machine`, hãy lấy giá trị nhỏ nhất, nếu không thì lấy giá trị nhỏ tiếp theo trong số chúng). Sau đó, độ dài bước thời gian cần tiến lên được tính toán dựa trên `next_time_on_machine` & thời gian hiện tại. Cuối cùng, thời gian tiếp theo của các máy có thời gian tiếp theo chậm hơn thời gian hiện tại sẽ được tiến lên thời gian hiện tại. Bộ phận nhả máy sẽ nhả các máy có thời gian nhả đạt đến thời gian hiện tại & cập nhật trạng thái của công việc & máy. Khi thao tác công việc hiện tại hoàn thành, máy đang được chiếm dụng sẽ trở nên nhàn rỗi & thao tác hiện tại của công việc này chuyển sang thao tác tiếp theo. Trạng thái của công việc & máy được cập nhật trong `assignable_job, job_on_machine`. Nếu tất cả thao tác của công việc hiện tại đã hoàn thành, công việc này sẽ không thể gán được & nếu máy cần cho thao tác tiếp theo bị chiếm dụng, công việc này vẫn không thể gán được.

- 3. A DRL scheduling framework for FJSP. In this sect, introduce overall DRL framework for FJSP, illustrated in Fig. 4: DRl framework for flexible job-shop scheduling problems. It is composed of:

1. state representation based on 2 state variables
2. PDR action space
3. reward function based on scheduling area
4. scheduling policy based on deep neural networks & a Softmax function
5. PPO agent with an actor-critic learning architecture.

– Khung lập lịch DRL cho FJSP. Trong phần này, giới thiệu khung DRL tổng thể cho FJSP, được minh họa trong Hình 4: Khung DRL cho các bài toán lập lịch xưởng linh hoạt. Khung này bao gồm:

1. biểu diễn trạng thái của mục dựa trên 2 biến trạng thái
2. Không gian hành động PDR
3. Hàm thưởng dựa trên vùng lập lịch
4. Chính sách lập lịch dựa trên mạng nơ-ron sâu & 1 hàm Softmax
5. Tác tử PPO với kiến trúc học tập tác nhân-phê bình.

- 3.1. State representation based on state variables. State representation depicts features of scheduling environment & determines size of state space, which is very crucial in RL scheduling methods. Various state representations for FJSP are presented in literature & are mainly focused on disjunctive graph [15, 22], feature matrix [24] & handcrafted state variables [8, 10, 14]. However, whether it is node features of disjunctive graph or matrix, or variable features, state features of job shop are mainly manually designed, which requires a large amount of processional domain knowledge, computing resources & time.

  – Biểu diễn trạng thái dựa trên biến trạng thái. Biểu diễn trạng thái mô tả các đặc điểm của môi trường lập lịch & xác định kích thước của không gian trạng thái, điều này rất quan trọng trong các phương pháp lập lịch RL. Nhiều biểu diễn trạng thái cho FJSP được trình bày trong tài liệu & chủ yếu tập trung vào đồ thị rời rạc [15, 22], ma trận đặc trưng [24] & các biến trạng thái thủ công [8, 10, 14]. Tuy nhiên, cho dù là đặc trưng nút của đồ thị rời rạc hay ma trận, hay đặc trưng biến, đặc trưng trạng thái của job shop chủ yếu được thiết kế thủ công, đòi hỏi lượng lớn kiến thức về miền xử lý, tài nguyên tính toán & thời gian.

  In this paper, a novel short state representation is proposed to reduce feature redundancy & to avoid handcrafted feature design & feature selection, which requires less computation time of environment state variables & scheduling policy networks. 2 state variables: `assignable_job, completed_op_of_job` are selected from Algorithm 1 as state features of job shop scheduling environment. Variable `assignable_job` is a Boolean vector to represent whether a job can be allocated or not. `completed_op_of_job` represents number of completed operation of a job & this value is then scaled by maximum number of operations in jobs to be in range $[0, 1]$. Length of both variables equals number of jobs. Finally, in order to represent environment state, scaled variables are simply concatenated to a vector whose length equals 2 times of number of jobs.

  – Trong bài báo này, 1 biểu diễn trạng thái ngắn mới được đề xuất để giảm sự dư thừa tính năng & để tránh thiết kế tính năng thủ công & lựa chọn tính năng, đòi hỏi ít thời gian tính toán hơn của các biến trạng thái môi trường & mạng chính sách lập lịch. 2 biến trạng thái: `assignable_job, completed_op_of_job` được chọn từ Thuật toán 1 làm các tính năng trạng thái của môi trường lập lịch job shop. Biến `assignable_job` là 1 vectơ Boolean để biểu diễn liệu 1 công việc có thể được phân bổ hay không. `completed_op_of_job` biểu diễn số lượng hoạt động đã hoàn thành của 1 công việc & giá trị này sau đó được chia tỷ lệ theo số lượng hoạt động tối đa trong các công việc nằm trong phạm vi $[0, 1]$. Độ dài của cả hai biến bằng số lượng công việc. Cuối cùng, để biểu diễn trạng thái môi trường, các biến được chia tỷ lệ chỉ cần được nối thành 1 vectơ có độ dài bằng 2 lần số lượng công việc.

  There are many advantages of our state representation:

  1. length of state features is much less than that in previous research which means less computation time of environment state variables & scheduling policy networks
  2. state features are unique in a scheduling solution, which suggests that state features can be easily distinguished by scheduling agent
  3. state features are directly derived from 2 state variables in Algorithm 1, avoiding massive experiments on feature design & selection.

– Biểu diễn trạng thái của chúng tôi có nhiều ưu điểm:

1. Độ dài của các đặc trưng trạng thái ngắn hơn nhiều so với nghiên cứu trước đây, đồng nghĩa với việc giảm thời gian tính toán các biến trạng thái môi trường & mạng lưới chính sách lập lịch.
2. Các đặc trưng trạng thái của mục là duy nhất trong 1 giải pháp lập lịch, điều này cho thấy các đặc trưng trạng thái có thể dễ dàng được phân biệt bởi tác nhân lập lịch.
3. Các đặc trưng trạng thái của mục được suy ra trực tiếp từ 2 biến trạng thái trong Thuật toán 1, tránh được các thử nghiệm lớn về thiết kế & lựa chọn đặc trưng.

- 3.2. Action space based on PDR. In DRl scheduling methods based on single agent, action is output of scheduling policy networks, which is usually an integer. Since FJSP needs to select a job & a machine at each decision, decompose this integer into 2 parts by dividing it by number of PDRs for machine selection where quotient is index of PDR for jobs assignment & remainder represents index of PDR for machine selection.

  – Không gian hành động dựa trên PDR. Trong các phương pháp lập lịch DRl dựa trên tác nhân đơn lẻ, hành động là đầu ra của mạng chính sách lập lịch, thường là 1 số nguyên. Vì FJSP cần chọn 1 công việc & 1 máy tại mỗi quyết định, hãy phân tích số nguyên này thành 2 phần bằng cách chia nó cho số PDR để chọn máy, trong đó thương là chỉ số của PDR để gán công việc & phần dư biểu thị chỉ số của PDR để chọn máy.

  In this paper, 6 PDRs are selected to construct action space for simplicity of implementation & ease of generalization. For selecting a job, 4 PDRs are selected directly from literature [27] including Shortest Processing (SPT), Most Work Remaining (MWRK), Most Operations Remaining (MOR) & Minimum ratio of Flow Due Date to Most Work Remaining (FDD/MWRK). Longest Remaining Machine time not including current operation processing time (LRM) is selected from [11] due to its excellent performance. First In First Out (FIFO) is selected because it is widely used in various scheduling problems. In addition, action space for selecting machine given a job is composed of SPT & Longest Processing Time (LPT). So that total size of action space equals number of PDRs for selecting jobs multiplied by number of PDRs for selecting machines. Definitions of these PDR are listed as follows:

  * SPT: $\min Z_{ij} = p_{ij}$
  * LPT: $\max Z_{ij} = p_{ij}$
  * FDD/MWKR: $\min Z_{ij} = \frac{\sum_1^j p_{ij}}{\sum_j^{n_i} p_{ij}}$
  * MOR: $\max Z_{ij} = n_i - j + 1$
  * LRM: $\max Z_{ij} = \sum_{j+1}^{n_i} p_{ij}$
  * FIFO: $\max Z_{ij} = t - Rt_i$

  where Z_{ij} is priority index of operation O_{ij}. p_{ij} is processing time of operation O_{ij}. n_i: number of operations for job J_i. j: number of completed operations. t: current time & Rt_i: release time of J_i.

  – Trong bài báo này, 6 PDR được chọn để xây dựng không gian hành động nhằm đơn giản hóa việc triển khai & dễ dàng khái quát hóa. Để lựa chọn 1 công việc, 4 PDR được chọn trực tiếp từ tài liệu [27] bao gồm Xử lý ngắn nhất (SPT), Công việc còn lại nhiều nhất (MWRK), Hoạt động còn lại nhiều nhất (MOR) & Tỷ lệ tối thiểu giữa Ngày đến hạn luồng & Công việc còn lại nhiều nhất (FDD/MWRK). Thời gian máy còn lại dài nhất không bao gồm thời gian xử lý hoạt động hiện tại (LRM) được chọn từ [11] do hiệu suất tuyệt vời của nó. Vào trước ra trước (FIFO) được chọn vì nó được sử dụng rộng rãi trong nhiều vấn đề lập lịch khác nhau. Ngoài ra, không gian hành động để lựa chọn máy cho 1 công việc bao gồm SPT & Thời gian xử lý dài nhất (LPT). Do đó, tổng kích thước của không gian hành động bằng số PDR để lựa chọn công việc nhân với số PDR để lựa chọn máy. Định nghĩa của các PDR này được liệt kê như sau:

  * SPT: $\min Z_{ij} = p_{ij}$
  * LPT: $\max Z_{ij} = p_{ij}$
  * FDD/MWKR: $\min Z_{ij} = \frac{\sum_1^j p_{ij}}{\sum_j^{n_i} p_{ij}}$
  * MOR: $\max Z_{ij} = n_i - j + 1$
  * LRM: $\max Z_{ij} = \sum_{j+1}^{n_i} p_{ij}$
  * FIFO: $\max Z_{ij} = t - Rt_i$

  trong đó Z_{ij} là chỉ số ưu tiên của phép toán O_{ij}. p_{ij} là thời gian xử lý của thao tác O_{ij}. n_i: số thao tác cho công việc J_i. j: số thao tác đã hoàn thành. t: thời gian hiện tại & Rt_i: thời gian giải phóng J_i.

- 3.3. Reward function based on scheduling area. In this paper, propose a comprehensible reward function based on scheduling area (1)

$$\text{reward}(s_t, a_t) = -p_{a,j} - \sum_M \text{vacancy}_m(s_t, s_{t+1})$$

  where processing time of allocated operations & time vacancy of all machines is computed after each dispatching action, where s_t: current state & s_{t+1} is next state after applying action a_t; a_t is jth operation of a job with processing time $p_{a,j}$; vacancy(s_t, s_{t+1}) is a function returning total time vacancy on machine set M while transitioning from state s_t to s_{t+1}.

  – Hàm thưởng dựa trên vùng lập lịch. Trong bài báo này, chúng tôi đề xuất 1 hàm thưởng dễ hiểu dựa trên vùng lập lịch (1)

$$\text{reward}(s_t, a_t) = -p_{a,j} - \sum_M \text{vacancy}_m(s_t, s_{t+1})$$

trong đó thời gian xử lý của các tác vụ được phân bổ & thời gian trống của tất cả các máy được tính sau mỗi hành động điều phối, trong đó s_t: trạng thái hiện tại & s_{t+1} là trạng thái tiếp theo sau khi áp dụng tác vụ a_t; a_t là tác vụ thứ j của 1 tác vụ có thời gian xử lý $p_{a,j}$; vacancy(s_t, s_{t+1}) là 1 hàm trả về tổng thời gian trống trên tập máy M khi chuyển từ trạng thái s_t sang s_{t+1}.

Proposed reward function is motivated by fact: total processing time of all jobs & time vacancy on all machines constructs scheduling area of all machines which equals maximum make-span multiplied by number of machines. As is demonstrated in Fig. 5: An example to show the scheduling area., scheduling area is composed of total processing time of all jobs (shaded area) & time vacancy on all machines (white color area). Relationship between accumulated reward & maximum scheduling makespan is derived from (2)

$$R = -a - b = -(a + b) = -S = -|M| * \text{makespan}$$

where R is accumulated reward, a: total processing time of all jobs, b: total time vacancy on all machines, S: scheduling time area & $|M|$: number of machines.

– Hàm phần thưởng được đề xuất dựa trên thực tế: tổng thời gian xử lý của tất cả các công việc & thời gian trống trên tất cả các máy tạo nên vùng lập lịch của tất cả các máy, bằng khoảng thời gian hoàn thành tối đa nhân với số máy. Như được minh họa trong Hình 5: Ví dụ minh họa vùng lập lịch., vùng lập lịch bao gồm tổng thời gian xử lý của tất cả các công việc (vùng tô bóng) & thời gian trống trên tất cả các máy (vùng màu trắng). Mối quan hệ giữa phần thưởng tích lũy & khoảng thời gian hoàn thành tối đa được suy ra từ (2)

$$R = -a - b = -(a + b) = -S = -|M| * \text{khongthigianhonthnh}$$

trong đó R là phần thưởng tích lũy, a: tổng thời gian xử lý của tất cả các công việc, b: tổng thời gian trống trên tất cả các máy, S: vùng thời gian lập lịch & $|M|$: số máy.

Obviously shown from (2): total reward & maximum makespan are negatively linearly dependent & coefficient is number of machines. I.e., minimizing maximum scheduling makespan is equivalent to maximizing total reward.

– Hiển nhiên từ (2): tổng phần thưởng & thời gian hoàn thành tối đa phụ thuộc tuyến tính âm & hệ số là số máy. Nghĩa là, việc giảm thiểu thời gian hoàn thành tối đa tương đương với việc tối đa hóa tổng phần thưởng.

- 3.4. Model training method based on PPO. In order to strengthen representation ability of RL scheduling method, scheduling policy is usually represented by DNNs e.g. CNN, RNN, & MLP. In our method, state feature vector is 1st fed to MLP to obtain a scalar score for each action & a Softmax function is then applied to output a distribution over computed score, which is shown in (3)

$$p(a_t|s_t) = \text{Softmax}(\text{MLP}_{\pi_\theta}(s_t)),$$

where $p(a_t|s_t)$ is selection probability of action a_t at time t in state s_t & θ is parameter of scheduling policy π. Structure of our scheduling policy is demonstrate in Fig. 4, which constructs actor network of PPO agent. Similar to actor network, critic network is implemented by MLP with 1 hidden layer. Parameters of our scheduling policy are learned by a clipped PPO whose loss function is expressed in (4)

$$L(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 \pm \epsilon)A_t)]$$

where $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, A_t is an estimator of advantage function at time step t, clip is a clipping function & ϵ is a hyper parameter which is used to limit boundary of objective function.

– Phương pháp huấn luyện mô hình dựa trên PPO. Để tăng cường khả năng biểu diễn của phương pháp lập lịch RL, chính sách lập lịch thường được biểu diễn bằng các DNN, ví dụ: CNN, RNN, & MLP. Trong phương pháp của chúng tôi, vectơ đặc trưng trạng thái đầu tiên được đưa vào MLP để thu được điểm số vô hướng cho mỗi hành động & sau đó áp dụng hàm Softmax để đưa ra phân phối trên điểm số đã tính toán, được thể hiện trong (3)

$$p(a_t|s_t) = \text{Softmax}(\text{MLP}_{\pi_\theta}(s_t)),$$

trong đó $p(a_t|s_t)$ là xác suất lựa chọn hành động a_t tại thời điểm t trong trạng thái s_t & θ là tham số của chính sách lập lịch π. Cấu trúc của chính sách lập lịch của chúng tôi được minh họa trong Hình 4, trong đó xây dựng mạng lưới tác nhân của tác tử PPO. Tương tự như mạng diễn viên, mạng phê bình được triển khai bằng MLP với 1 lớp ẩn. Các tham số của chính sách lập lịch của chúng tôi được học bởi 1 PPO bị cắt xén có hàm mất mát được biểu thị trong (4)

$$L(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 \pm \epsilon)A_t)]$$

trong đó $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, A_t là 1 ước lượng của hàm lợi thế tại bước thời gian t, clip là 1 hàm cắt xén & ϵ là 1 siêu tham số được sử dụng để giới hạn biên của hàm mục tiêu.

Detailed training process is provided by Algorithm 2: Model training method based on PPO. Training process includes 2 aspects: data collection & policy learning. When collecting training data, T independent complete trajectories of scheduling are generated & they are collected in memory buffer M. On policy learning stage, training data are used for K times. At each time, these data are randomly divided into batches whose length is related to scale of instances & scheduling agent learns from each batch data. After replaying these experience samples evenly, prioritized experience replay is performed for

C times. Training process will stop until episode reaches maximum iterations or result is convergent or scheduling is time out. Define: result is thought as convergent if makespan values are same in 30 decision steps & training time is limited in an hour.

– Quá trình đào tạo chi tiết được cung cấp bởi Thuật toán 2: Phương pháp đào tạo mô hình dựa trên PPO. Quá trình đào tạo bao gồm 2 khía cạnh: thu thập dữ liệu & học chính sách. Khi thu thập dữ liệu đào tạo, T quỹ đạo hoàn chỉnh độc lập của lập lịch được tạo & chúng được thu thập trong bộ đệm M. Ở giai đoạn học chính sách, dữ liệu đào tạo được sử dụng trong K lần. Tại mỗi thời điểm, những dữ liệu này được chia ngẫu nhiên thành các lô có độ dài liên quan đến quy mô của các trường hợp & tác nhân lập lịch học từ mỗi dữ liệu lô. Sau khi phát lại các mẫu trải nghiệm này 1 cách đồng đều, việc phát lại trải nghiệm được ưu tiên sẽ được thực hiện trong C lần. Quá trình đào tạo sẽ dừng lại cho đến khi tập đạt số lần lặp tối đa hoặc kết quả hội tụ hoặc lập lịch hết thời gian. Định nghĩa: kết quả được coi là hội tụ nếu các giá trị makespan giống nhau trong 30 bước quyết định & thời gian đào tạo bị giới hạn trong 1 giờ.

- 4. Experiments. In this sect, in order to show effectiveness of our DRL scheduling environment for FJSP (Algorithm 1) & to evaluate performance of proposed DRL scheduling methods, a group of experiments are performed on public FJSP benchmarks with various sizes & results are compared with different types of scheduling methods from recent literature. Finally, training details e.g. training time are demonstrated.

  – Trong phần này, để chứng minh tính hiệu quả của môi trường lập lịch DRL cho FJSP (Thuật toán 1) & đánh giá hiệu suất của các phương pháp lập lịch DRL được đề xuất, 1 nhóm các thử nghiệm được thực hiện trên các chuẩn FJSP công khai với nhiều kích thước khác nhau & kết quả được so sánh với các loại phương pháp lập lịch khác nhau từ các tài liệu gần đây. Cuối cùng, các chi tiết đào tạo, ví dụ như thời gian đào tạo, sẽ được trình bày.

  - 4.1. Benchmark instances & baseline models. In this paper, 2 well-known benchmarks of FJSP are used to evaluate our proposed methods including MK instances (MK01-MK10) in [4] & 3 group of LA instances (edata, rdata, & vdata each with 40 instances) in [12].

    – Trong bài báo này, 2 chuẩn mực nổi tiếng của FJSP được sử dụng để đánh giá các phương pháp đề xuất của chúng tôi bao gồm các thể hiện MK (MK01-MK10) trong [4] & 3 nhóm thể hiện LA (edata, rdata, & vdata mỗi nhóm có 40 thể hiện) trong [12].

    This paper compared with PDR, exact solver, meta-heuristic & DRL models. 6 PDRs are used to compare scheduling results where 4 out of 6 PDRs are compared in old scheduling environment & our proposed environment. Also compare with well-known Google OR-Tools which is a powerful constraint programming solver showing strong performance in solving industrial scheduling problems. For meta-heuristic scheduling methods, 2 recent improved Genetic Algorithms are selected from [18] & [5]. For DRL scheduling methods, compared with competing models in recent 3 years, including 4 DRl methods proposed by Feng et al.[9], Zeng et al.[26], Song et al.[22] & Lei et al.[15] resp., GSMA model [13] which is a MARL scheduling method & obtains state-of-art results, DANILE model [23] which is based on attention mechanism.

    – Bài báo này so sánh với PDR, bộ giải chính xác, mô hình meta-heuristic & DRL. 6 PDR được sử dụng để so sánh kết quả lập lịch trong đó 4 trong số 6 PDR được so sánh trong môi trường lập lịch cũ & môi trường đề xuất của chúng tôi. Cũng so sánh với Google OR-Tools nổi tiếng, 1 bộ giải lập trình ràng buộc mạnh mẽ cho thấy hiệu suất cao trong việc giải quyết các vấn đề lập lịch công nghiệp. Đối với các phương pháp lập lịch meta-heuristic, 2 Thuật toán di truyền được cải tiến gần đây được chọn từ [18] & [5]. Đối với các phương pháp lập lịch DRL, so sánh với các mô hình cạnh tranh trong 3 năm gần đây, bao gồm 4 phương pháp DRl do Feng et al.[9], Zeng et al.[26], Song et al.[22] & Lei et al.[15] đề xuất, tương ứng, mô hình GSMA [13] là 1 phương pháp lập lịch MARL & thu được kết quả tiên tiến, mô hình DANILE [23] dựa trên cơ chế chú ý.

  - 4.2. Model configurations. PPO agent adopts actor-critic architecture where actor networks represent scheduling policy & critic networks calculate state-value of policy networks. Actor network is implemented by MLP & Softmax function while critic network is only represented by MLP. Both networks are optimized by Adam optimizer, use ReLU as activation function & have only 1 hidden layer with dynamic hidden dimension which equals length of state features. For each problem size, train policy network for $\leq$ 8000 iterations, each of which contains 9 independent trajectories (i.e., complete scheduling process of instances) & use a dynamic batch size which equals 2 times of scale (total number of operations of all jobs) of an instance. For PPO, set epochs of updating network to 10 & clipping parameter epsilon to 0.2. Set discount factor γ to 0.999 & learning rate are $1e-3, 3e-3$ for actor & critic network resp. For prioritized experience replay, parameter α is set to 0.6, value of β anneals from 0.4 to 1, number of prioritized experience replay C is set to 1, & number of convergence training steps is 2000.

    – Tác nhân PPO áp dụng kiến trúc actor-critic trong đó các mạng actor biểu diễn chính sách lập lịch & các mạng critic tính toán giá trị trạng thái của các mạng policy. Mạng actor được triển khai bởi hàm MLP & Softmax trong khi mạng critic chỉ được biểu diễn bởi MLP. Cả hai mạng đều được tối ưu hóa bởi trình tối ưu hóa Adam, sử dụng ReLU làm hàm kích hoạt & chỉ có 1 lớp ẩn với chiều ẩn động bằng với độ dài của các đặc trưng trạng thái. Đối với mỗi kích thước vấn đề, hãy huấn luyện mạng policy cho $\leq$ 8000 lần lặp, mỗi lần lặp chứa 9 quỹ đạo độc lập (tức là quy trình lập lịch hoàn chỉnh của các trường hợp) & sử dụng kích thước lô động bằng 2 lần quy mô (tổng số thao tác của tất cả các công việc) của 1 trường hợp. Đối với PPO, đặt các kỷ nguyên cập nhật mạng thành 10 & tham số cắt epsilon thành 0,2. Đặt hệ số chiết khấu γ thành $0,999$ & tốc độ học là $1e-3, 3e-3$ tương ứng với mạng actor & critic. Đối với phát lại trải nghiệm được ưu tiên, tham số α được đặt thành 0,6, giá trị của β được ủ từ 0,4 đến 1, số lần phát lại trải nghiệm được ưu tiên C được đặt thành 1, & số bước đào tạo hội tụ là 2000.

  - 4.3. Results analysis. In order to evaluate our proposed scheduling environment, 1st test performance of 6 PDRs in our proposed scheduling environment on MK benchmark instances where 6 PDRs are used to select a job while SPT is used for

selection of machines. Results are shown in Table 1: Scheduling results of PDRs on MK benchmark instances. Widely-used 4 PDRs (SPT, MWKR, FIFO, MOR) are performed in our environment & results are compared with that in old scheduling environment where their best results are selected from literature [22]. Symbol "-" means: specific results are not recorded in literature. Best scheduling result of all 6 PDRs on each instance are recorded in minPDR row. Besides, results of 2 DRL scheduling methods proposed resp. by Feng et al.[9] & Zeng et al.[26] are also compared in Table 1 & their results are directly from their literature instead of our implementation environment.

– Để đánh giá môi trường lập lịch đề xuất của chúng tôi, hiệu suất thử nghiệm đầu tiên của 6 PDR trong môi trường lập lịch đề xuất của chúng tôi trên các phiên bản chuẩn MK trong đó 6 PDR được sử dụng để chọn 1 công việc trong khi SPT được sử dụng để chọn máy. Kết quả được hiển thị trong Bảng 1: Kết quả lập lịch của PDR trên các phiên bản chuẩn MK. 4 PDR được sử dụng rộng rãi (SPT, MWKR, FIFO, MOR) được thực hiện trong môi trường của chúng tôi & kết quả được so sánh với kết quả trong môi trường lập lịch cũ, trong đó kết quả tốt nhất của chúng được chọn từ tài liệu [22]. Ký hiệu "-" có nghĩa là: kết quả cụ thể không được ghi lại trong tài liệu. Kết quả lập lịch tốt nhất của tất cả 6 PDR trên mỗi phiên bản được ghi lại trong hàng minPDR. Bên cạnh đó, kết quả của 2 phương pháp lập lịch DRL do Feng et al.[9] & Zeng et al.[26] đề xuất cũng được so sánh trong Bảng 1 & kết quả của chúng được lấy trực tiếp từ tài liệu của họ thay vì môi trường triển khai của chúng tôi.

As shown in table 1, results indicate: performance of SPT, FIFO, MOR in our environment is improved while performance of MWKR is suddenly worse than before. LRM obtained best average results among all 6 PDRs & even better than some DRL scheduling methods e.g. models proposed by Feng et al. & Zeng et al., which is surprisingly interesting. Best result on different instances is distributed in different PDRs, e.g. MWKR obtained best results on MK1 & FIFO performs best on MK3. So that, minPDR can get better results than LRM.

– Như thể hiện trong bảng 1, kết quả cho thấy: hiệu suất của SPT, FIFO, MOR trong môi trường của chúng tôi được cải thiện trong khi hiệu suất của MWKR đột nhiên kém hơn trước. LRM đạt được kết quả trung bình tốt nhất trong số tất cả 6 PDR & thậm chí còn tốt hơn 1 số phương pháp lập lịch DRL, ví dụ như các mô hình do Feng & cộng sự đề xuất & Zeng & cộng sự, điều này thật đáng ngạc nhiên. Kết quả tốt nhất trên các trường hợp khác nhau được phân phối trong các PDR khác nhau, ví dụ: MWKR đạt được kết quả tốt nhất trên MK1 & FIFO hoạt động tốt nhất trên MK3. Do đó, minPDR có thể đạt được kết quả tốt hơn LRM.

2nd group of experiments are also performed on MK benchmark instances to evaluate generality of our proposed DRL scheduling agent in our environment. Train these instances independently for 5 times & average results are recorded in "Ours" column. As shown in Table 2, compare performance of our proposed DRL scheduling model with exact solver (OR-Tools), meta-heuristic scheduling methods (2SGA[18] & SLGA [5]), DRL scheduling methods (GMAS[13], DANILE[23], & models proposed by Song et al.[22]) & minPDR method. LB & UB columns resp. record lower bound & upper bound scheduling results in literature. Results in these compared methods are directly from literature except results of PDRs which are performed in our environment.

– Nhóm thí nghiệm thứ 2 cũng được thực hiện trên các trường hợp chuẩn MK để đánh giá tính tổng quát của tác nhân lập lịch DRL được đề xuất của chúng tôi trong môi trường của chúng tôi. Đào tạo các trường hợp này độc lập trong 5 lần & kết quả trung bình được ghi lại trong cột "Của chúng tôi". Như thể hiện trong Bảng 2, hãy so sánh hiệu suất của mô hình lập lịch DRL được đề xuất của chúng tôi với bộ giải chính xác (OR-Tools), các phương pháp lập lịch meta-heuristic (2SGA[18] & SLGA [5]), các phương pháp lập lịch DRL (GMAS[13], DANILE[23], & các mô hình do Song et al.[22] đề xuất) & phương pháp minPDR. Các cột LB & UB tương ứng ghi lại kết quả lập lịch giới hạn dưới & giới hạn trên trong tài liệu. Kết quả trong các phương pháp được so sánh này được lấy trực tiếp từ tài liệu, ngoại trừ kết quả của PDR được thực hiện trong môi trường của chúng tôi.

Average makespan is usually used to evaluate accuracy of scheduling solutions. As demonstrated in Table 2: Makespan performance of our DRL model on MK benchmark instances, GMAS which is a DRL scheduling method based on MARL, obtained best average result. Nevertheless, they used optimal results of each instance rather than average results. Average makespan of our proposed DRL model is smaller than SLGA, DANILE, Song, & minPDR methods & is larger than OR-Tools, 2SGA & GMAS methods, which shows: performance of DRL scheduling methods are getting close to meta-heuristic & exact solver. However, different from complex scheduling networks of GMAS, our model is simple & easy to train to converge, which is stable.

– Makespan trung bình thường được sử dụng để đánh giá độ chính xác của các giải pháp lập lịch. Như được minh họa trong Bảng 2: Hiệu suất Makespan của mô hình DRL của chúng tôi trên các trường hợp chuẩn MK, GMAS, 1 phương pháp lập lịch DRL dựa trên MARL, đã thu được kết quả trung bình tốt nhất. Tuy nhiên, họ đã sử dụng kết quả tối ưu của từng trường hợp thay vì kết quả trung bình. Makespan trung bình của mô hình DRL đề xuất của chúng tôi nhỏ hơn các phương pháp SLGA, DANILE, Song, & minPDR & lớn hơn các phương pháp OR-Tools, 2SGA & GMAS, điều này cho thấy: hiệu suất của các phương pháp lập lịch DRL đang tiến gần đến trình giải meta-heuristic & chính xác. Tuy nhiên, khác với các mạng lập lịch phức tạp của GMAS, mô hình của chúng tôi đơn giản & dễ huấn luyện để hội tụ, điều này rất ổn định.

Beside, evaluate convergence performance of our DRL scheduling method. As is demonstrated in Fig. 6: Taining trajectories & training time on MK benchmarks., our DRL model on all MK instances is convergent in half an hour on average & number of needed trajectories is < 900. Convergence time of 8 out of 10 instances is < 600 seconds, which is close to industrial time limitation.

– Ngoài ra, hãy đánh giá hiệu suất hội tụ của phương pháp lập lịch DRL của chúng tôi. Như được minh họa trong Hình 6: Định vị quỹ đạo & thời gian huấn luyện trên chuẩn MK., mô hình DRL của chúng tôi trên tất cả các trường hợp MK đều hội tụ trung bình trong nửa giờ & số quỹ đạo cần thiết là < 900. Thời gian hội tụ của 8 trên 10 trường hợp là < 600 giây, gần với giới hạn thời gian công nghiệp.

In order to show stability & generality of our proposed DRL scheduling agent in our environment, more experiments are performed on LA benchmark instances, including edta, rdata, & vdata, whose flexibility is getting increased. As shown in Table 3: Average makespan comparison LA instances, compare with various scheduling methods in literature e.g. exact solver OR-Tools, meta-heuristic 2SGA, DRL scheduling methods: DANILE, GMAS, Lei [15] & Song as well as PDR methods. Results of these methods are directly from literature. Results of PDR methods are from running in our proposed environment & only minimum result of 6 PDRs on each instance is recorded in minPDR column.

– Để chứng minh tính ổn định & tính tổng quát của tác nhân lập lịch DRL được đề xuất trong môi trường của chúng tôi, nhiều thử nghiệm hơn đã được thực hiện trên các phiên bản chuẩn LA, bao gồm edta, rdata, & vdata, với tính linh hoạt ngày càng tăng. Như được thể hiện trong Bảng 3: So sánh makespan trung bình của các phiên bản LA, hãy so sánh với các phương pháp lập lịch khác nhau trong tài liệu, ví dụ: bộ giải chính xác OR-Tools, meta-heuristic 2SGA, các phương pháp lập lịch DRL: DANILE, GMAS, Lei [15] & Song cũng như các phương pháp PDR. Kết quả của các phương pháp này được trích dẫn trực tiếp từ tài liệu. Kết quả của các phương pháp PDR được thực hiện trong môi trường được đề xuất của chúng tôi & chỉ có kết quả tối thiểu là 6 PDR trên mỗi phiên bản được ghi lại trong cột minPDR.

As is demonstrated in Table 3, our proposed DRL scheduling agent got smaller average makespan than DANILE, Lei, Song, & PDR models & obtained larger average makespan than OR-Tools, GMAS, & 2SGA, which shows competing performance of our proposed DRL scheduling agent in our environment. More interestingly, minPDR obtained smaller makespan than Lei model which is used to solve large-scale dynamic FJSP. That shows efficiency of our proposed environment again.

– Như được minh họa trong Bảng 3, tác nhân lập lịch DRL đề xuất của chúng tôi có makespan trung bình nhỏ hơn các mô hình DANILE, Lei, Song, & PDR & đạt makespan trung bình lớn hơn OR-Tools, GMAS, & 2SGA, điều này cho thấy hiệu suất cạnh tranh của tác nhân lập lịch DRL đề xuất trong môi trường của chúng tôi. Thú vị hơn, minPDR đạt makespan nhỏ hơn mô hình Lei, được sử dụng để giải các bài toán FJSP động quy mô lớn. Điều này 1 lần nữa cho thấy hiệu quả của môi trường chúng tôi đề xuất.

Finally, training time of our proposed DRL scheduling agent on edata, rdata, & vdata are depicted in Fig. 7: Training time on edata, rdata, & vdata instances. Our proposed DRL model can be trained to converge in an hour on all benchmark instances & total training time on edta, rdata, & vdata are 22064, 31650 & 40137 seconds, resp., which shows training time rises as increase of complicity of scheduling instances.

– Cuối cùng, thời gian đào tạo của tác nhân lập lịch DRL được đề xuất của chúng tôi trên edata, rdata, & vdata được mô tả trong Hình 7: Thời gian đào tạo trên các phiên bản edata, rdata, & vdata. Mô hình DRL được đề xuất của chúng tôi có thể được đào tạo để hội tụ trong 1 giờ trên tất cả các phiên bản chuẩn & tổng thời gian đào tạo trên edta, rdata, & vdata tương ứng là 22064, 31650 & 40137 giây, cho thấy thời gian đào tạo tăng lên khi mức độ phức tạp của các phiên bản lập lịch tăng lên.

- 5. Conclusion. In this paper, proposed a chronological discrete event simulation based DRL environment for FJSP. Scheduling process is described by a simulation algorithm where state changes are captured by state variables & reward function is calculated based on scheduling area of machines at each decision step. In this simulaiton environment, an end-to-end DRL framework is proposed based on actor-critic PPO, providing a flexible infrastructure for design of state representation, action space & scheduling policy networks. Besides, a simple DRL model for FJSP is presented by defining state representation of very short state features based on 2 state variables in simulation environment, action space composed of widely-used priority dispatching rules in literature & scheduling policy baesd on MLP with only 1 hidden layer. Various experiments are performed on classic benchmark instances with different sizes & results show: performance of PDR is improved in our environment, even better than some DRL methods. Besides, our DRL scheduling method provides competing scheduling performance compared with DRL, meta-heuristic & PDR methods.

  – Bài báo này đề xuất 1 môi trường DRL dựa trên mô phỏng sự kiện rời rạc theo trình tự thời gian cho FJSP. Quá trình lập lịch được mô tả bằng 1 thuật toán mô phỏng, trong đó các thay đổi trạng thái được ghi lại bởi các biến trạng thái & hàm thưởng được tính toán dựa trên diện tích lập lịch của máy tại mỗi bước quyết định. Trong môi trường mô phỏng này, 1 khuôn khổ DRL đầu cuối được đề xuất dựa trên PPO tác nhân-phê bình, cung cấp 1 cơ sở hạ tầng linh hoạt cho việc thiết kế biểu diễn trạng thái, không gian hành động & mạng lưới chính sách lập lịch. Bên cạnh đó, 1 mô hình DRL đơn giản cho FJSP được trình bày bằng cách định nghĩa biểu diễn trạng thái của các đặc trưng trạng thái rất ngắn dựa trên 2 biến trạng thái trong môi trường mô phỏng, không gian hành động bao gồm các quy tắc phân bổ ưu tiên được sử dụng rộng rãi trong tài liệu & chính sách lập lịch dựa trên MLP với chỉ 1 lớp ẩn. Nhiều thử nghiệm khác nhau được thực hiện trên các phiên bản chuẩn cổ điển với các kích thước khác nhau & kết quả cho thấy: hiệu suất của PDR được cải thiện trong môi trường của chúng tôi, thậm chí còn tốt hơn 1 số phương pháp DRL. Hơn nữa, phương pháp lập lịch DRL của chúng tôi cung cấp hiệu suất lập lịch cạnh tranh so với các phương pháp DRL, meta-heuristic & PDR.

  Future research will mainly focus on design of scheduling policy networks as well as state representation. State features can be thought as texts or images so that various networks in NLP & CV fields can be used in scheduling policy networks, e.g. SPP networks, TextCNN, & Transformer.

  – Nghiên cứu trong tương lai sẽ chủ yếu tập trung vào thiết kế mạng lưới chính sách lập lịch cũng như biểu diễn trạng thái. Các đặc điểm trạng thái có thể được hiểu dưới dạng văn bản hoặc hình ảnh, do đó, các mạng lưới khác nhau trong trường NLP & CV có thể được sử dụng trong mạng lưới chính sách lập lịch, ví dụ: mạng SPP, TextCNN, & Transformer.

3 Miscellaneous