國立臺北科技大學

資訊工程系碩士班

碩士學位論文

以深度強化學習求解動態機台數量的彈性零工式工廠排程問題

Learning to Dispatch for Flexible Job-Shop Scheduling with Dynamic Number of Machines via Deep Reinforcement Learning

研究生：張禹弘

指導教授：劉建宏 博士、尤信程 博士

中華民國一百一十二年七月

國立臺北科技大學
研究所碩士學位論文口試委員會審定書

本校 資訊工程 研究所 張禹弘 君

所提論文，經本委員會審定通過，合於碩士資格，特此證明。

學位考試委員會

委　　員：尤信程

黃乙玄

劉建宏

指導教授：尤信程 劉建宏

所　　長：劉建宏

中 華 民 國　一百零一十二　年　七　月　十九　日

摘要

論文名稱：以深度強化學習求解動態機台數量的彈性零工式工廠排程問題

頁數：六十一頁

校所別：國立臺北科技大學 資訊工程系碩士班

畢業時間：一百一十一學年度 第二學期

學位：碩士

研究生：張禹弘

指導教授：劉建宏 博士、尤信程 博士

關鍵詞：深度強化學習、彈性零工式工廠排程

在現代的生產管理領域中，動態彈性零工式工廠排程（Dynamic Flexible Job-Shop Problem, DFJSP）是經常遇到的問題。隨著產品生產線的發展越來越複雜，機台有可能突然發生故障或者是需要休息，因此我們需要設計一個動態調度框架來處理隨著時間而產生機台數量變化的不確定性。

在本論文中，主要研究的方向為如何產生一個合理且良好的排程，以得到最短完工時間（Makespan）。我們使用深度強化學習代理自動學習優先調度規則，將 DFJSP 以分離圖（Disjunctive Graph）的形式來表示，並利用圖神經網路將分離圖嵌入到狀態中以供代理學習，因此策略網路的大小不會隨著機台數量而變動，可具有處理動態數量的能力，並且能夠有效地實現對大規模實例的泛化。實驗結果表示，與現有最佳優先調度規則演算法相比表現出更強大的性能，驗證了所提出框架的有效性。

ABSTRACT

Title: Learning to Dispatch for Flexible Job-Shop Scheduling with Dynamic Number of Machines via Deep Reinforcement Learning

Pages: 61

School: National Taipei University of Technology

Department: Computer Science & Information Engineering

Time: July, 2023

Degree: Master

Researcher: Yu-Hung Chang

Advisor: Chien-Hung Liu, Ph.D., Shing-Chern You, Ph.D.

Keywords: Deep Reinforcement Learning, Flexible Job-Shop Scheduling Problem

In the field of modern production management, the Dynamic Flexible Job-Shop Problem (DFJSP) is a frequently encountered issue. As the production lines for products become increasingly complex, machines may experience sudden failures or require breaks. Therefore, we need to design a dynamic scheduling framework to handle the uncertainty of machine quantity changes over time.

In this thesis, our focus is on generating a reasonable and well-performing schedule to minimize the Makespan, which is the total time required to complete all jobs. We utilize a deep reinforcement learning agent to automatically learn priority scheduling rules. The DFJSP is represented in the form of a disjunctive graph, and we embed the disjunctive graph into the state using graph neural networks for the agent's learning. This ensures that the size of the policy network remains constant regardless of the number of machines, enabling it to handle dynamic quantities and effectively generalize to large-scale instances. Experimental results demonstrate the effectiveness of the proposed framework compared to existing state-of-the-art priority scheduling algorithms.

致謝

非常感謝劉建宏教授與尤信程教授在兩年碩士期間的努力指導，讓我在此期間學習到了許多研究方法，在研究陷入瓶頸時，教授也給了許多的建議，讓我能夠順利的完成研究。

再來感謝實驗室的同學們，楷融、創晨、昱安、孜頤、峻暐和維俊，在這兩年期間我們互相鼓勵扶持，遇到問題時會互相討論，讓我能夠有討論的對象以及一起努力的同伴。

最後感謝我的父母，您們支持我就讀研究所，尊重我的想法，讓我得以專心於研究，順利完成碩士生涯，再次感謝您們的付出。

目錄

表目錄

圖目錄

第一章 緒論

1.1 研究動機

零工式工廠排程問題（Job-Shop Scheduling Problem，簡稱 JSSP）是電腦科學（Computer Science）和作業研究（Operations Research）中著名的組合優化問題。它普遍存在於各個行業，比如製造業和運輸業[1]。JSSP 是將一組工作中的每個任務（操作）按照特定的順序在多台機器上完成，要考慮如何分配機器已達到效益最大化，通常我們會以最短完工時間（Makespan）當作目標去最小化它。這類型問題已被證實為 NP-Hard 問題，因此想要找到 JSSP 的一組最佳解是非常具有挑戰性，在學術領域中，大多會使用啟發式演算法來解決，例如基因演算法（Genetic Algorithm，簡稱 GA）。

在現代的生產管理領域中，動態彈性零工式工廠排程問題（Dynamic Flexible Job-Shop Problem，簡稱 DFJSP）是一個具有挑戰性和現實意義的問題。它是由 JSSP 衍伸出來。DFJSP 比起 JSSP 多了動態變化和更加彈性的條件約束，因此較具有通用性。傳統的排程方法無法應對現實生產環境中的不確定性和動態變化，導致排程效果不佳。隨著製造業的發展和全球化競爭的加劇，生產線的複雜性和變動性不斷提高，因此需要一種靈活且高效的排程方法來應對這種變化。

首先，現代生產線往往由大量的機台和工作組成，這些機台可能會突然發生故障或需要休息，導致工作無法順利進行，影響工廠的生產效率和產能利用率。因此，我們需要一種能夠處理動態數量機台的排程方法，以確保在機台故障或休息時仍然能夠有效地安排工作。

其次，傳統的排程方法通常會使用優先調度規則（Priority Dispatching Rule，簡稱 PDR）來進行排程規劃，PDR 以其計算效率、易於實施以及處理實際調度系統中固有的不確定性的能力而聞名。在文獻中[2]實驗了大量的 PDR，有幾種 PDR 在 JSSP 和動態加入工作的問題中得到較好的 Makespan。不過基於事先定義好的優先調度規則在不同實例中可能會有很大差異，設計一個有效的 PDR 可能是一個耗時的過程，需要擁有領

域知識以及不斷進行嘗試，才能定製出合適的 PDR，這樣會增加人力成本和排程的複雜性。因此，需要一種能夠自動學習和調整優先調度規則的方法，以適應生產線的變化和優化排程效果。

近年來深度強化學習（Deep Reinforcement Learning，簡稱 DRL）技術的最新進展在解決各種組合優化問題方面顯示出前景。在[1]提出的方法中，使用 DRL 在 JSSP 得到良好的表現，在[3-8]提出的方法中，可以使用 DRL 解決具有動態問題的 JSSP 和 DFJSP，不過其處理的動態問題多以隨機工作到達為主，缺少動態增減機台數量的問題。因此，我們想探索如何利用 DRL 自動化設計 PDR 來解決以隨機工作到達為主的 DFJSP，從而提高其有效性、效率和泛化能力。

1.2 研究目的

本研究的主要目的是開發一種基於深度強化學習的動態調度框架，以最小化 DFJSP 中的 Makespan。具體來說，我們將探索如何使用深度強化學習代理來自動學習優先調度規則，以提高排程的效率和準確性。

首先，我們將 DFJSP 建模為分離圖（Disjunctive Graph）的形式，其中每個工作和機台都表示為圖中的節點，而工作之間的先後順序和機台之間的條件約束則表示為圖中的邊。這種表示方式能夠清晰地描述工作和機台之間的關係，為後續的排程優化提供準確的資訊。

其次，我們將使用圖神經網絡（Graph Neural Network，簡稱 GNN）使分離圖嵌入到狀態中。GNN 是一種強大的機器學習方法，可以處理圖結構化數據。通過將分離圖嵌入到狀態中，代理可以從圖中獲取有關工作和機台之間的關係和條件約束的資訊，更好地理解和處理排程問題。

我們的研究將基於深度強化學習的框架，使用強化學習代理來學習最佳的優先調度規則。代理通過與環境的互動，學習如何選擇最佳的工作順序和機台分配，以最小化 Makespan。我們將使用深度強化學習算法 Proximal Policy Optimization（PPO）來訓練代

理，通過對現有最佳 PDR 演算法進行實驗比較，我們將驗證所提出的動態調度框架的有效性。我們預計所提出的方法能夠實現更好的 Makespan，提高生產線的效率和靈活性。此外我們還將評估所提出框架在大規模實例上的泛化能力，以確保其在實際應用中的可行性和可靠性。

最終，我們期望本研究的成果能夠為組合優化領域提供一種靈活且高效的動態調度方法，改善生產線的排程效果，減少生產成本，提高產能利用率，並為企業提供更具競爭力的優勢。同時，我們的研究也將推動深度強化學習組合優化領域的應用，探索其在解決其他相關排程問題上的潛力。我們認為本研究的成果能對組合優化領域的發展做出貢獻，促進製造業的持續發展和創新。

1.3 論文架構

本論文的架構如下：第二章為背景介紹與相關研究，將介紹排程問題的類型和動態規則的定義，並介紹現有使用深度強化學習來解決排程問題的 L2D 架構；第三章會提出對 L2D 的改進方法，使其可以處理動態機台數量的排程問題；第四章為本論文的各種實驗結果，針對研究目的來提出與其相關的研究問題，設計不同的實驗來評估我們的方法的有效程度，最後根據實驗結果進行討論；第五章為結論與未來研究方向，將本論文的研究結果做出總結，並提出未來可以繼續研究的方向。

第二章 背景介紹與相關研究

本章會介紹本論文中相關領域背景，介紹排程問題中 JSSP、FJSP、DFJSP 的差別，並定義本論文中要解決的問題。接著會介紹其他人所提出的深度學習框架，包括本論文中會採用到的 L2D 架構，探討該架構的有效性。

2.1 背景介紹

本節將介紹常見排程問題類型，並針對與我們本研究中要處理的排程類型加以介紹，認識現有幾種強大的優先調度規則。

2.1.1 常見排程問題類型

排程問題可以分為多個類型，每種類型都有其特定的限制和目標，有著其獨特的特徵和挑戰。排程中有三種基本元件：工作（Job）、任務（Task）、機器（Machine），每一台機器可以處理一個任務。在其他論文中，有時會將任務改稱為作業（Operation）。在本論文中，任務亦即作業，機器亦即機台。

排程可以分成兩大類型：單階作業（Single Operation）和多階作業（Multiple Operation），底下可再細分多種類型。以下是一些常見的排程問題類型：

- 單階作業：每一個工作包含單一任務。
  - 單機排程（Single Machine Scheduling）：

    最為簡單的一種排程問題，此問題中有包含多個工作，但只有一台機器，因此只需要考慮工作的優先順序即可。
  - 平行機排程（Parallel Machine Scheduling）：

    包含多個工作和多台機器，每一台機器可以處理任何的任務。與單機排程的差別僅在於任務可分配機器數量變多。
- 多階作業：每一個工作包含多個任務，任務之間的約束關係可再細分成以下排程。
  - 流線式工廠排程（Flow Shop Scheduling）：

每一個工作中，任務之間的操作順序相同。舉例來說，有一工作中包含 A、B、C 三個任務，機器只能先從 A 開始處理，接著是 B，最後是 C。其他的工作都是相同 A→B→C 的順序，可以看作是工廠中的流水線，任務順序只有一種。

- 零工式工廠排程（Job Shop Scheduling）：

  每一個工作中都有獨特的任務操作順序，擁有較高的排程複雜度，學術界主要以此作為研究對象。彈性零工式工廠排程（Flexible Job-Shop Problem）和動態彈性零工式工廠排程（Dynamic Flexible Job-Shop Problem）皆從此問題延伸。稍後於 2.1.2 小節和 2.1.3 小節再詳細介紹。

- 開放式工廠排程（Open Shop Scheduling）：

  開放式工廠排程與零工式工廠排程相似，差別在於任務操作順序是否以任意決定的。比如任務順序為 A→B→C，那麼 A→C→B 或 B→A→C 都是可行的，因此相較於零工式工廠排程，條件約束比較寬鬆。

常見排程問題類型的樹狀圖如圖 2-1 所示。

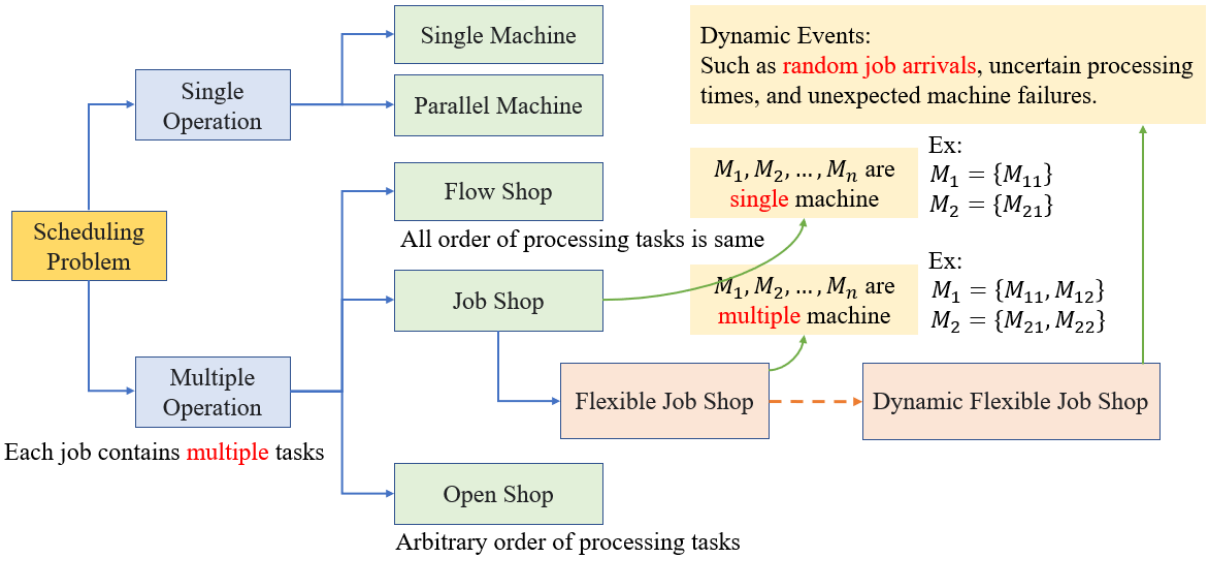

圖 2-1 常見排程問題類型的樹狀圖

2.1.2 零工式工廠排程

零工式工廠排程（Job-Shop Scheduling Problem，簡稱 JSSP）的定義如下：一個標準 JSSP 實例由一組工作 $J = \{J_1, J_2, \ldots, J_n\}$ 和一組機器 $M = \{M_1, M_2, \ldots, M_m\}$ 所組成，每一個工作 J_i 都包含多個任務 O_{ij}，其中 $1 \leq j \leq m_i$，m_i 表示為 J_i 需要經過 m 台機器來處理任務，O_{ij} 處理時間表示為 $p_{ij} \in \mathbb{N}$。一個 JSSP 實例大小表示為 $|J| \times |M|$。JSSP 有三點約束條件如下列所示：

1. 每一個任務 O_{ij} 都需要由指定的一台機器 $k \in M$ 來處理。
2. 任務 $\{O_{i1}, O_{i2}, \ldots, O_{m_i}\}$ 必須按照順序處理，表示為 $O_{i1} \rightarrow O_{i2} \rightarrow \cdots \rightarrow O_{m_i}$，通常會將此稱為「優先約束」（precedence constraints）。
3. 每台機器在任何時刻中只能處理一個任務，機器工作時間不會重疊。當該機器開始處理任務後便無法中斷，必須要處理完當前任務才能繼續處理其他任務。

為了解決一個 JSSP 實例，需要找到所有 O_{ij} 的開始時間 S_{ij} 並最小化 Makespan。Makespan 可以表示為 $C_{max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$，其中 C_{ij} 為 O_{ij} 的結束時間。目標函數是最小化 C_{max} 並且要能滿足上述三點約束條件。

我們可以使用分離圖（Disjunctive Graph）來表示一個 JSSP 的實例，如圖 2-2 所示。

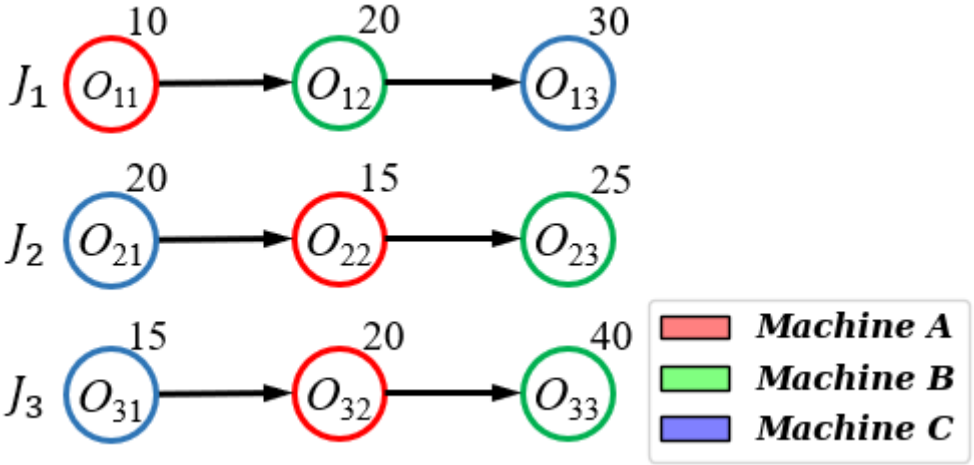

圖 2-2 JSSP 實例

分離圖是學術界常用來表示 JSSP 的一種圖形，它能明確顯示任務和機器之間的處理順序，並且能用於顯示一個 JSSP 實例和其解決方案（簡稱為解）。圖 2-2 是一個 3×3 的 JSSP 實例，共有 3 個工作和 3 台機器，每一個工作都包含 3 個任務。每一個節點代表一個任務 O_{ij}，節點圓圈的顏色代表需要由對應顏色的機器來完成，節點右上角的數字代表 O_{ij} 處理時間 p_{ij}。節點之間的黑色有向箭頭代表優先約束，每一個工作的任務必須按照順序處理。

圖 2-3 是一個 JSSP 實例的解。其中分離圖$G = (O, C, D)$，O 是節點的集合；C 是合取（conjunction）的集合，對應圖中實線有向邊；D 是析取（disjunction）的集合，對應圖中虛線有向邊。conjunction 表示同一個工作內任務之間的優先約束，而 disjunction 表示機器在每個任務之間處理的順序。想要找到一個 JSSP 的解，等同於確定相同機器中每兩個節點之間的「箭頭方向」，之後便能依照分離圖計算出 Makespan。

要計算 Makespan，首先需要知道每一個任務的開始時間 S_{ij}。以下我們將以圖 2-3 其中幾個任務做為計算範例。首先，所有機器的開始時間為 0，當機器 A 處理 O_{11} 時，開始時間 $S_{11} = 0$，結束時間 $C_{11} = S_{11} + p_{11} = 10$。當機器 C 處理 O_{21} 時，必須要先讓機器 C 處理完 O_{31}，因此 O_{21} 的結束時間 $C_{21} = S_{21} + p_{21} = C_{31} + p_{21} = 35$。當機器 A 處理 O_{22} 時，必須等 O_{21} 和 O_{32} 處理完（對應於 conjunction 和 disjunction），因此 O_{22} 的結束時間 $C_{22} = S_{22} + p_{22} = \max(C_{21}, C_{32}) + p_{22} = 50$。

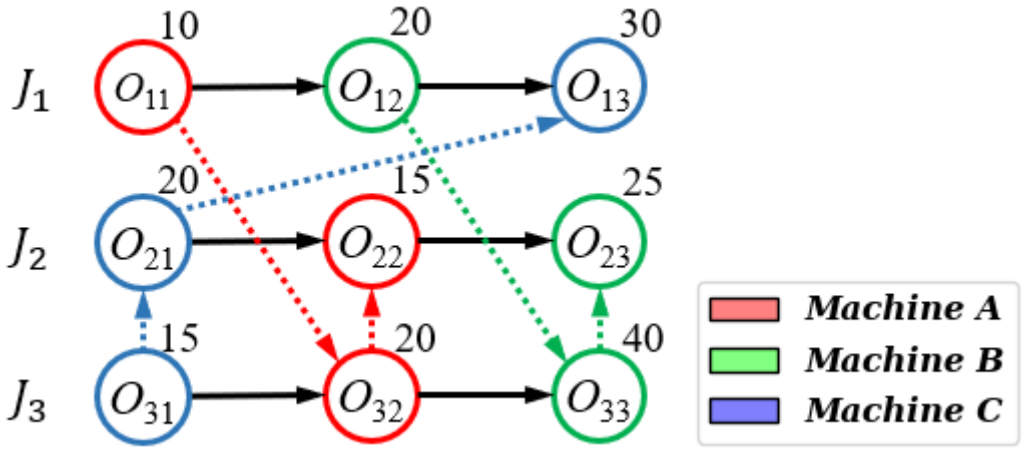

圖 2-3 基於分離圖的 JSSP 實例的一個解

將所有開始時間 S_{ij} 計算完後，可以得到 Makespan $= \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\} = 100$。除了用分離圖表示 JSSP 的解，還可以使用甘特圖來表示工作任務和機器之間的排程狀況，如圖 2-4 所示。

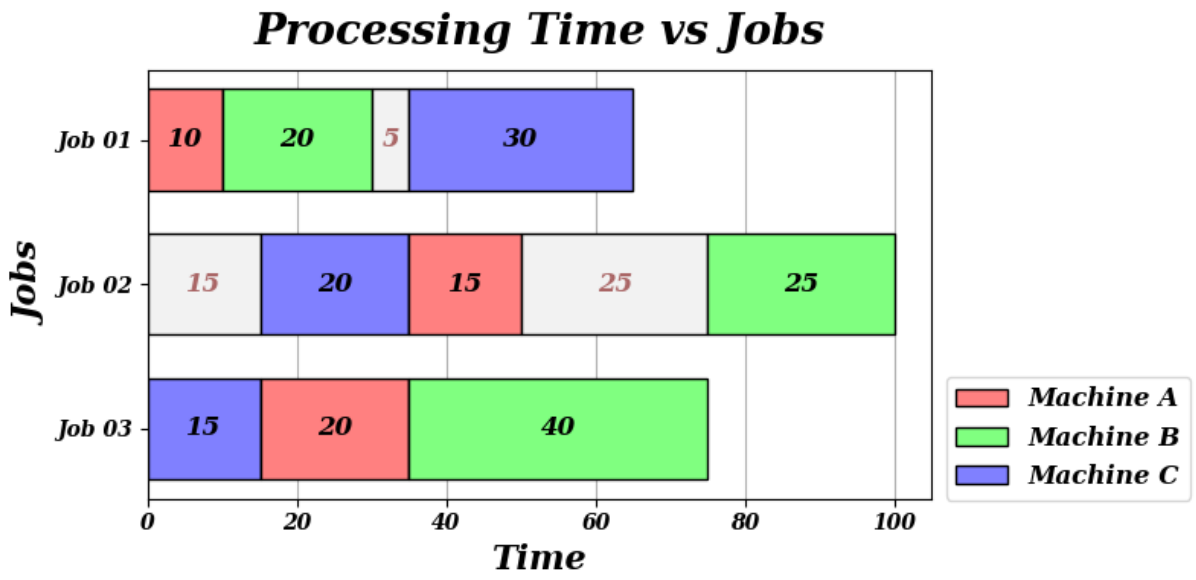

圖 2-4 基於甘特圖的 JSSP 實例的一個解

從上圖來看，可以得知各工作的每個任務之間存在著一些未排程時間，如 Job 01 在第 2 個任務結束後等待 5 個時間單位，才執行第 3 個任務，這是因為此時機器 C 正在處理其他任務，因此該任務需要等待機器 C 完成當前任務才能開始，所以存在著空閒時間。除此之外，能清楚看到機器在不同工作的處理時間不會重疊。當 Job 02 在處理完第 3 個任務後，整體排程才算結束，因此 Makespan 為當下的結束時間 100。

2.1.3 動態彈性零工式工廠排程

我們先介紹彈性零工式工廠排程（Flexible Job-Shop Problem，簡稱 FJSP），與 JSSP 的差別在於，從一組單一機器變成一組機器集合，每個任務都能從對應的機器集合當中選擇一台機器處理。依照本論文的要處理的問題，我們定義一組機器集合 M 包含多種類型機器，即 $M = \{M_1, M_2, \dots, M_m\}$。而每種機器類型 M_k 包含多台相同類型的機器，即 $M_k = \{M_{k1}, M_{k2}, \dots, M_{kl_k}\}$，其中 l_k 表示在機器類別 M_k 上擁有的機器數量。

動態彈性零工式工廠排程（Dynamic Flexible Job-Shop Problem，簡稱 DFJSP）是在 FJSP 的基礎下添加動態事件，本論文中只要提到具有動態事件的 FJSP 或是具有動態機台數量的 FJSP，即意指 DFJSP。當然 JSSP 也可以添加動態事件，但本論文是以 DFJSP 做為研究對象，因此將對 DFJSP 做說明。

動態事件有很多種，根據問題的需要來決定動態類型。例如作業隨機到達、機器故障、工作取消、緊急工作插入、交貨日期或加工時間變化等[6]。現實中，排程方案應根據生產情況的變化不斷調整，因此添加動態事件的排程問題更具有研究價值。

在本論文中，我們想要解決的問題是「機台數量」的動態增減，即每隔一段時間就會發生機器增加或減少的事件，在單位時間內隨機事件發生次數的機率分布服從卜瓦松分布（Poisson distribution）。我們定義每種機器類型 $M_k = \{M_{k1}, M_{k2}, \dots, M_{kl_k}\}$，其中 l_k 會受到動態影響而變化，$l_k = \{1, 2, \dots, n_k\}$。 n_k 表示機器類型 M_k 擁有的機器數量上限。關於卜瓦松分布和增減機器的數量公式，請詳見 4.2 節。

2.1.4 優先調度規則

優先調度規則（Priority Dispatching Rule）是用於生產排程和任務優先順序安排的規則，可以幫助決定哪些任務應該先執行，例如先到先服務（First-Come-First-Serve，簡稱 FCFS）。不過像 FCFS 這種簡單且常見的 PDR 在某些情況下可能存在效能問題，因此以下我們針對現有幾個強大的 PDR 並進行介紹：

- 最短處理時間（Shortest Processing Time，簡稱 SPT）：

  SPT 是一個常見且簡單的優先調度規則，根據工作所需的處理時間，優先選擇處理時間最短的工作。這可以最大程度地減少任務的平均等待時間，並提高生產效率。

- 最早截止日期（Earliest Due Date，簡稱 EDD）：

  EDD 是基於工作的截止日期而制定的優先調度規則。按照截止日期，優先選擇截止日期最早的工作進行處理。這有助於確保重要或急迫的工作能夠在截止日期之前完成。在我們的問題中，工作沒有設定截止日期，因此 EDD 在本論文無法使用。

- 最早開始日期（Earliest Start Time，簡稱 EST）：

  EST 是根據工作的預計開始日期而制定的優先調度規則。根據工作的預計開始日期，優先選擇最早可開始的工作進行處理，有助於確保工作在計劃時間內開始進行。

- 最多剩餘工作（Most Work Remaining，簡稱 MWKR）：

  MWKR 是根據工作剩餘未完成的任務處理時間總和而制定的優先調度規則。優先選擇剩餘處理時間較多的工作進行處理，有助於確保不會留下處理時間較長的工作。在 JSSP 的實驗中表現好[1]。

- 流程到期日與最多剩餘工作的最小比率（Flow Due Date per Most Work Remaining，簡稱 FDD/MWKR）：

  FDD/MWKR 是根據工作已完成的任務處理時間總和，再加上工作到達的時間，最後除以工作剩餘未完成的任務處理時間總和而制定成的優先調度規則，有助於優先處理累計完成任務時間較少的工作。在 JSSP 的實驗中表現好[1]。

- 最多剩餘操作（Most Operation Remaining，簡稱 MOPNR）：

  MOPNR 是根據工作剩餘未完成的任務個數總和而制定的優先調度規則。優先選擇剩餘任務個數較多的工作進行處理，有助於確保不會留下剩餘任務較多的工作。在 JSSP 的實驗中表現好[1]。

  關於以上 PDR 的縮寫和公式，可以在以下網站的 Appendix 中獲得資訊：

  https://www.projectmanagement.ugent.be/research/machine_scheduling

2.2 相關研究

本節將介紹現有用於解決排程問題的強化學習架構，以及用於求出問題最佳解的約束編程求解器，並介紹圖神經網路（Graph neural network，簡稱 GNN）在解決排程問題上有哪些優勢。

2.2.1 有使用強化學習的方法

Jens Heger 等人[3] 提出利用 Q-learning 來處理動態工作到達時間和改變產品組合，儘管在未曾見過的場景中，也能夠給出可行的結果。Yakup Turgut 等人[4] 提出利用 deep Q-network（DQN）來解決具有動態工作到達的 DJSP，代理人需要決定從等待的任務中選擇其中一個任務並將其發送到第一台機器進行處理，並以最小化任務的延遲時間做為目標函數，證明了比 PDR 中的 SPT 和 EDD 的性能還要好。Jingru Chang 等人[6] 提出利用 double deep Q-networks（DDQN）解決隨機工作到達的 DFJSP，並統整了用於動態調度問題的現有 9 種 RL 方法。從文獻中收集各個學者提出的方法[6, 7]來看，研究主要集中於動態 JSSP 和 DFJSP，動態問題以隨機工作到達為主，而非本論文要解決的動態問題，能夠了解到動態機台數量是比較少學者在進行的研究。Cong Zhang 等人[1] 提出 L2D 架構（Learning to Dispatch），使用 GNN 和 PPO 來學習 JSSP。它提供了一個新穎的概念，以往是以工作和機器做為狀態空間，來讓 RL 代理分配工作。而 L2D 直接從調度狀態空間的表示中進行學習，實現了一種 end-to-end 的方式學習策略。在[7, 9]中有以 L2D 的概念提出新的方法來解決動態問題。Kun Lei 等人[7, 8] 將強化學習代理分成高層和低層，使用 DDQN 作為高層代理，緩存新到達的工作，並決定要分配哪些工作給低層，亦即將 DFJSP 分割成靜態的 FJSP 並交由低層代理來處理。低層代理使用 GIN 對分離圖進行編碼，並利用 PPO 來學習策略。低層代理共有兩個，其中一個代理決定要處理的工作，另一個代理則是決定要分配給哪一台機器。Yi Zhang 等人[5] 提出一種基於多代理技術的分佈式調度架構，避免中央伺服器的工作量太大，不利於解決大規模問題和頻繁擾動。使用基於 PPO 演算法的調度模型自學習策略，對加入新工作、機器故障等不

可預測的事件進行處理。以早於預定的時間量、工作負載平衡、產品利潤做為目標函數，在時間、成本、資源利用率等方面實現全面且優越的表現。

從上述現有的方法中，我們想以 Cong Zhang 等人[1] 提出的 L2D 方法做為基礎，將其擴展為能處理具有動態機台數量的 FJSP，以實現本論文要達成的目標。雖然 Yi Zhang 等人[5] 提供能處理機器故障的方法，但是不選擇的原因是：

1. 架構複雜且沒有提供程式碼，難以實現和擴展。
2. 實驗的問題不是通用的 DFJSP，不好與其他強化學習方法比較。

而 L2D 有提供程式碼，並且架構簡單，易於擴展，學術上有論文[7]以此架構為基礎，擴展成能處理不同類型問題的方法，因此我們認為 L2D 較為適合做為本論文使用。

如何擴展 L2D，具體上我們會從狀態模型、特徵值、動作模型、獎勵函數、方法中的步驟進行修改，因此首先我們要驗證在各個學者主要研究的隨機工作到達之動態問題，使用 L2D 是否能夠有效的解決，並且在泛化能力（generalization）上，是否能夠以較小尺寸的問題實例訓練出來的模型直接應用於大尺寸的問題實例上。

表 2-1 為我們在具有隨機工作到來的動態事件下，使用 L2D 方法處理 DJSSP 的效能。OR-Tools 在提出 L2D 方法的論文中有被使用，它是用於求出問題最佳解的求解器。表 2-1 用已訓練模型在不同實例大小上的比較，其測試集中的工作數量一開始只有一半，每經過一段隨機時間後增加一個新工作，直至工作數量變為初始數量的 2 倍時停止增加新工作。L2D 和 OR-Tools 欄底下的數值為 Makespan，Gap 欄底下的數值為 L2D 和 OR-Tools 的 Makespan 差距。測試集實例數量為 600 筆，OR-Tools 在求解每一筆實例限制時間為 60 秒。原始論文中，L2D 和 OR-Tools 在各種不同實例大小下，差距落在 10.9% 到 39.5%，使用小尺寸實例訓練出來的模型用於預測大尺寸實例的差距落在 9.4% 到 26.5%。從我們的實驗結果中來看，差距皆符合在合理範圍內，因此我們可以確定 L2D 方法也能夠處理隨機工作到達的 JSSP，並且能夠以較小尺寸的問題實例訓練出來的模型直接應用於大尺寸的問題實例上。

表 2-1 L2D 在隨機工作到來的 JSSP 之效能表現

實例大小		L2D	OR-Tools	Gap (Our)
模型	測試集			
15x15	20x15	1865	1462	27.6%
	50x20	3644	3084	18.2%
	100x20	6322	5962	6.1%
20x20	20x15	1862	1458	27.6%
	50x20	3662	3074	19.1%
	100x20	6444	5972	7.9%
30x20	20x15	1866	1466	27.3%
	50x20	3653	3086	18.4%
	100x20	7620	5971	27.7%

雖然 L2D 可以處理隨機工作到來的 JSSP，但以現在的 L2D 方法並不能處理具有動態機台數量的 FJSP，因為 L2D 只能處理單一機器（受到 JSSP 的限制），所以在第三章中會詳細介紹我們的方法。在下一節中，我們將介紹 L2D 架構，以便了解 L2D 的運作狀況。

2.2.2 L2D 架構

L2D 主要分成兩個部分，狀態嵌入（state embedding）和學習策略（learning policy），狀態嵌入會使用圖神經網路（Graph neural network）中的圖同構網路（Graph Isomorphism Network，簡稱 GIN）作為狀態嵌入網路，而學習策略會選擇 Actor-Critic 架構中的 PPO 來進行學習，整體架構如圖 2-5 所示。

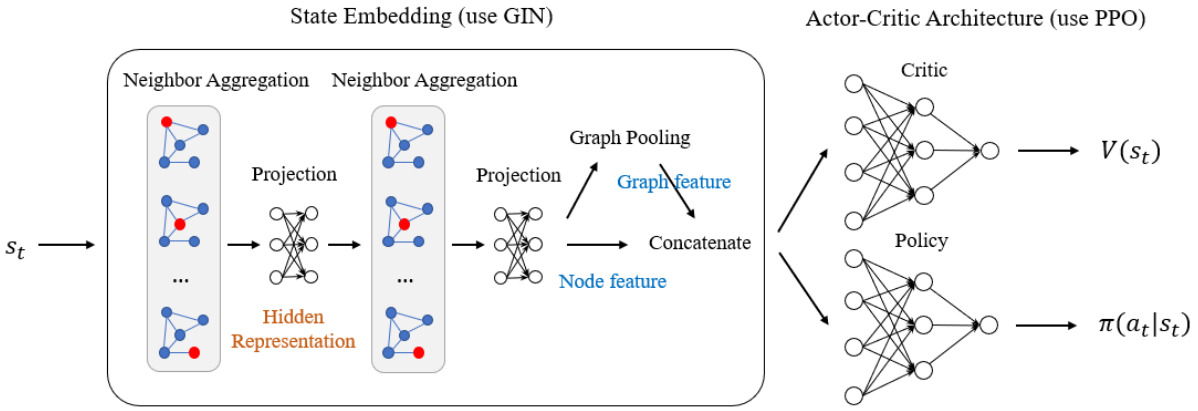

圖 2-5 L2D 整體架構

想要讓將深度強化學習應用於 JSSP，需要將 JSSP 描述成馬可夫決策過程（Markov decision process，簡稱 MDP），此部分將在 2.2.4 小節說明，GNN 的詳細介紹將在 2.2.3 小節說明，本小節僅大致講述架構中每一個步驟的功能。

狀態嵌入一詞是指將狀態向量透過神經網路映射（mapping）到另一組狀態向量。PPO 的 Policy 是指策略網路，用於輸入當前的狀態向量來產生當前要選擇的動作。

GNN 泛指專門用於處理圖形（Graph）數據的神經網路架構，GIN 是 GNN 的其中一種架構，就如同 AlexNet 和卷積神經網絡（Convolutional Neural Network，簡稱 CNN）之間的關係。在 L2D 中，狀態是帶有特徵值的分離圖，因此會選擇擅長處理圖形數據的 GIN 作為狀態嵌入網路，它可以輸入不固定數量的節點和邊，也可以輸入不固定的圖形形狀，但每一個節點必須要有相同數量的資訊（特徵）。

GIN 並非讓圖形節點變成神經網路中的神經元，神經網路的尺寸大小還是固定的，

而是以圖形的鄰接矩陣（adjacency matrix）和每一個節點的特徵值（features）作為神經網路的輸入，將圖形的資訊映射到特徵向量中。如果輸入的兩個圖形不是同構的（isomorphism），則產生的兩組特徵向量就會不一樣，以便後續讓 PPO 能根據相似的圖形學習到相似的動作策略。

Neighbor Aggregation 可以讓圖形每一個節點具有相鄰節點的資訊。Projection 為 multilayer perceptron（MLP）結構，用以獲取節點的隱藏表示。Graph Pooling 是將整個圖形壓縮成一個節點，使該節點具有整個圖形的資訊。Concatenate 是將 Projection 所輸出具有各個節點的資訊與經過 Graph Pooling 所得到的整個圖形資訊做串接，使特徵向量包含節點特徵和圖形特徵。輸入 GIN 的圖形節點數量多寡不會影響輸出特徵向量的長度，因此能使用於各種神經網路的輸入。最後 GIN 輸出固定長度的特徵向量會被學習策略網路當作輸入特徵，進行各種學習任務。學習策略可以使用任一種 Actor-Critic 架構，而在原始論文中是使用強化學習的 Proximal Policy Optimization（PPO），PPO 是一種相對高效穩定的策略優化演算法。它使用近端策略優化的方法，通過限制策略更新的幅度來確保學習的穩定性。這種控制策略更新的方式有助於避免過大的更新，從而緩解了演算法中的變動性問題，提高了訓練的穩定性。PPO 在處理連續動作空間時表現良好，對於需要在連續動作空間中學習的問題，PPO 可以有效地學習和優化連續動作策略。它使用樣本軌跡的機率比率來進行策略更新，這使得它能夠在連續動作空間中進行準確且有效的策略搜索。

PPO 在應用於大規模問題或分散式執行時也具有良好的可擴展性。它的策略更新步驟可以並行執行，並且可以與分佈式計算框架相結合，以加速訓練過程。而 PPO 也是一種通用演算法，可與各種神經網路架構一起使用。因此將 PPO 作為學習策略的方法是一個不錯的選擇。

2.2.3 圖神經網路和卷積神經網路的比較

在一些以圖像（picture）作為神經網路輸入的任務當中，通常會結合卷積神經網路進行狀態嵌入，後續再將輸出的特徵向量輸入進強化學習框架進行訓練。CNN 的主要優勢之一是它們能夠從原始數據中提取局部和全局的特徵。通過卷積層，CNN 能夠在圖像中捕捉局部結構和模式，並進行特徵提取。然後，池化層可以降低特徵的尺寸和計算負擔，同時保持重要資訊。

這種層次化的結構使得 CNN 在視覺任務中表現出色，如圖像分類、物體檢測和語義分割等，不過這種方法對於 L2D 並不適合。L2D 是以 JSSP 的分離圖作為輸入，分離圖是由節點（node）和邊（edge）所組成的圖形（Graph），其特徵是每個節點具有不固定數量的鄰居節點，這會導致無法使用 CNN 的卷積運算（Convolution）。

圖像屬於歐幾里德結構數據（Euclidean Structure Data），資料格式排列整齊，對於某個節點，可以很容易找出其鄰居節點，或者是相同的鄰居節點數量。而圖形是屬於非歐幾里德結構數據。

Graph neural network（GNN）擅長處理以圖形作為輸入的任務，比如社交網絡、分子結構等，在生活中也有許多應用程式以圖形的形式表示數據，比如電子商務系統或是論文引用網絡。GNN 通過在圖形上聚合和傳播訊息，捕捉圖形中節點之間的結構（局部結構）和整張圖形的特徵（全局結構）。聚合運算（Aggregation）是指將來自相鄰節點的訊息組合在圖形結構中，使得 GNN 獲取節點之間的關係資訊。如圖 2-6 所示。

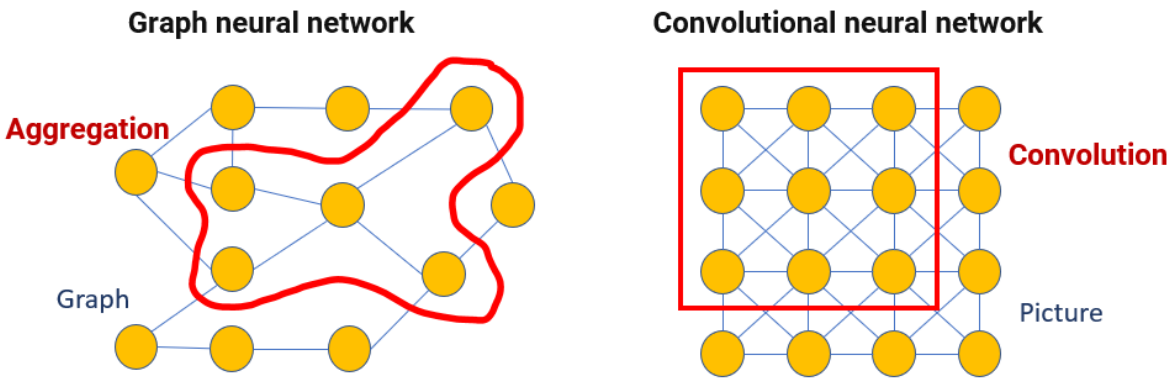

圖 2-6 聚合運算與卷積運算的差別

聚合有 sum、mean、max 等運算方法。聚合類似於 CNN 中的卷積，都是逐個對局部元素進行運算，差別在於 CNN 中每個元素具有相同數量的鄰居元素，而 GNN 則是不固定的。由於 GNN 的特性，GNN 能夠在推論、分類、鏈接預測和圖形生成等任務上取得良好的結果。

GNN 底下不只有 GIN 這一種架構，為何不使用圖卷積網路（Graph Convolutional Network，簡稱 GCN）當作狀態嵌入網路？首先考慮到狀態嵌入的目的，一般來說，我們輸入同樣的圖形，應該要映射到一組相同的特徵向量，才能對後續學習策略有幫助。那麼圖同構是否與圖形相等是一樣的？在圖論的觀點下，兩個同構的圖形會被當作同一個圖形來研究。如圖 2-7 所示，左右兩個圖形的節點擁有相同鄰居，因此這兩個圖形是同構的。

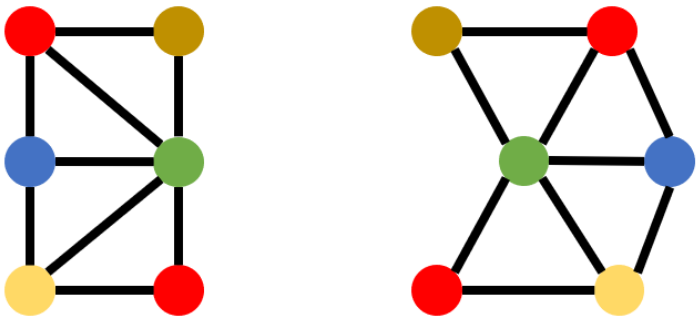

圖 2-7 圖同構的示例

GIN 對於圖形的節點排列順序不敏感。這意味著即使節點的排列不同，GIN 也能夠產生相同結果的狀態嵌入。而 GCN 在一定程度上依賴於節點的局部鄰居。GCN 通過局部卷積操作對節點的鄰居特徵進行聚合。這會導致 GCN 對於節點排列順序敏感，同一個圖形的不同節點排列可能導致不同的嵌入結果。

此外 GIN 具有較強的表示能力（Representation）。它可以捕捉更豐富的圖形結構和節點特徵，這有助於更好地嵌入圖形的狀態。GIN 通過多層全連接網絡以及自適應的加權操作，能夠更全面地整合全局和局部的圖形資訊。自適應的加權操作是指在圖形嵌入過程中使用可學習的權重對節點的特徵進行聚合，即圖 2-5 中的 Projection。GCN 在處理簡單的圖形結構和節點特徵時表現良好。然而，當處理較複雜的圖形結構和節點之間的遠距離關係時，GCN 會無法充分捕捉全局圖形資訊，導致節點表示的資訊丟失或模糊。

因此 GIN 比起 GCN 更適合作為本論文中的狀態嵌入網路。

2.2.4 L2D 馬可夫決策過程公式

根據馬可夫決策過程公式，我們需要定義狀態（state）、動作（action）、選擇動作後的狀態轉移（state transition）以及獎勵（reward），以下將分別介紹：

- 狀態

定義狀態 s_t，其中決策步驟 $t = \{0, 1, \dots, T\}$，T 表示決策步驟數量上限。由於我們將分離圖做為狀態做為輸入，因此 s_0 表示一個 JSSP 實例的分離圖 $G = (O, C)$，O 是節點的集合；C 是合取的集合。s_T 表示一個 JSSP 的完整解決方案 $G = (O, C, D)$，D 是析取的集合。s_t 表示一個 JSSP 解的當前狀態 $G = (O, C, D(t))$，其中 $D(t) \subseteq D$，表示在決策步驟 t 下已經分配好的析取的邊，即確定了相同機器中每兩個節點之間的有向邊。對於每個節點 $o \in O$，都會擁有兩個特徵。第一個特徵是該任務完成時間的下限，第二個是特徵是二元指標 $I(o, s_t) \in \{0, 1\}$，當 o 在 s_t 之前的狀態已被排程，則為 1，否則為 0。圖 2-8 顯示了一個 JSSP 的 s_0, s_t, s_T，節點上方的格子為該節點的特徵。

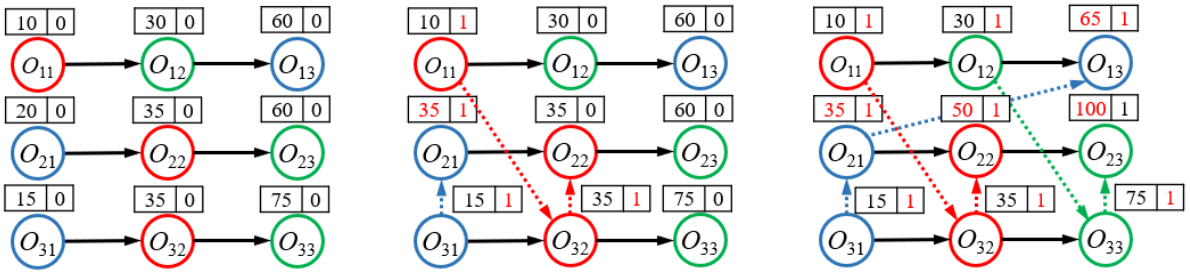

圖 2-8 一個 JSSP 的初始狀態、當前狀態、中止狀態

- 動作

定義動作 $a_t \in A_t$，a_t 為決策步驟 t 下的一個可行任務。動作空間 A_t 會根據當前狀態 s_t 而有所變化。可行的任務會受到優先約束的影響，如果前面任務還沒被選擇過，則後面任務就是不可行的。圖 2-9 表示一個 JSSP 在特定狀態下的動作空間，假設在狀態 s_3 時，O_{11}, O_{31}, O_{32} 已被選擇過，那麼之後不再出現於動作空間中。可行任務為每一個工作已完成任務的下一個待做任務，如圖中黃色背景的節點。而白色背景的節點在當前狀態下屬於不可行任務，因此 $A_3 = \{O_{12}, O_{21}, O_{33}\}$。

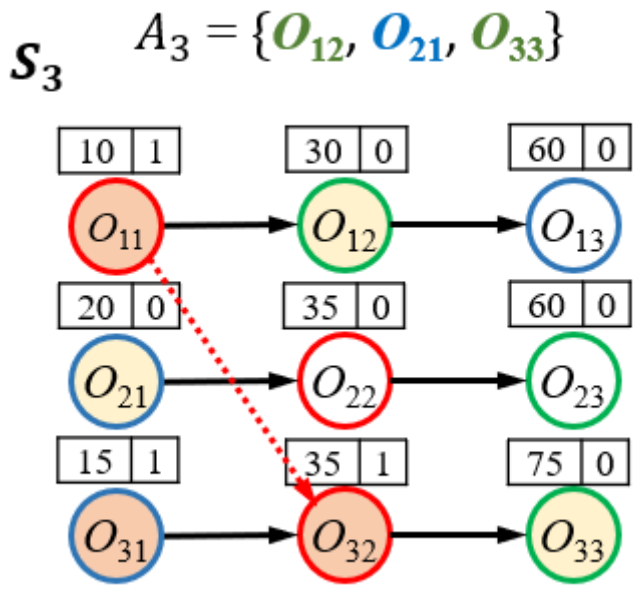

圖 2-9 一個 JSSP 在特定狀態下的動作空間

- 狀態轉移

在狀態 s_t 下選擇下一步要調度的任務 a_t，狀態 s_t 會轉移成 s_{t+1}。除了會影響到狀態的分離圖中的析取集合，每一個節點的特徵也會隨之變化，如圖 2-10 所示。當選擇 $a_3 = O_{21}$ 時，機器 C 會從上一個完成的任務 O_{31} 切換到 O_{21} 開始處理，因此 O_{31} 和 O_{21} 之間添加一條析取的邊，並且 O_{21} 以及其優先約束以後的任務 O_{22}, O_{23} 都會更新第一個特徵，即該任務完成時間的下限 $C_{LB}(O_{ij},\ s_t) = r_{ij} + p_{ij}$，若 O_{ij} 等同於 a_t，則 r_{ij} 等同該任務開始時間 S_{ij}，否則 r_{ij} 等同於 $O_{i(j-1)}$ 的結束時間 $C_{i(j-1)}$。a_3 的第二個特徵將被設置成 1。

- 獎勵

目標是學習如何一步一步地進行調度，使 Makespan 最小化。因此獎勵函數定義為 $R(a_t, s_t) = H(s_t) - H(s_{t+1})$，其中 $H(s_t)$ 是 Makespan 下限，即 $\max_{i,j}\{C_{LB}(O_{ij},\ s_t)\}$。以圖 2-11 做為範例，$s_3$ 和 s_4 的 Makespan 下限皆為 75，因此在 s_3 的狀態下選擇動作 a_3 的獎勵值為 0。

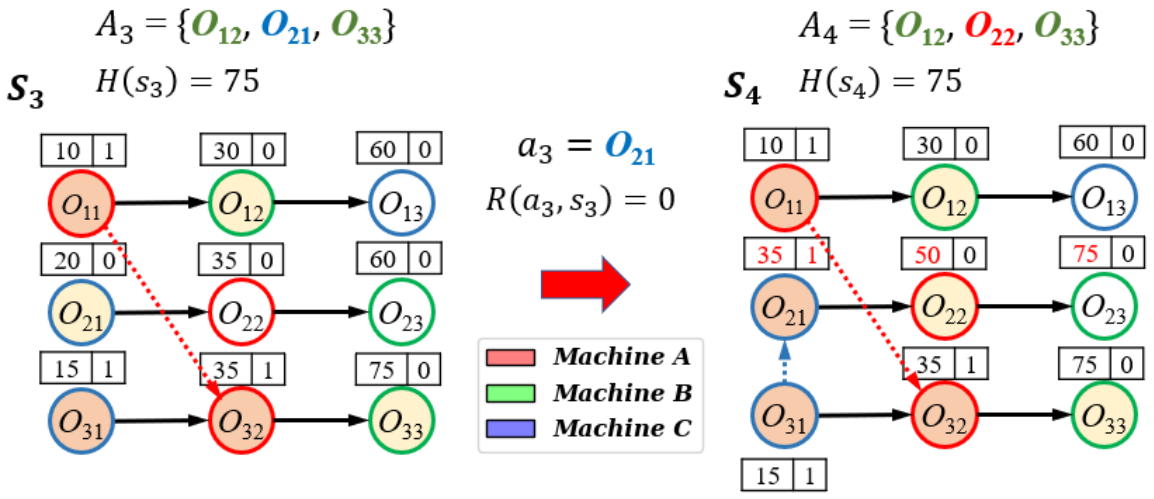

圖 2-10 一個 JSSP 的狀態轉移

2.2.5 約束編程求解器

約束編程求解器可以解決如 JSSP 等靜態的排程問題，只要給予足夠時間運算就能求出問題最佳解。即使是具有動態事件的 DFJSP，只要我們事先確定好動態事件何時發生，將其轉換成靜態問題，就可以使用約束編程求解器進行求解。

OR-Tools 是 Google 開發的求解器，名字中的 OR 表示 Operations Research（作業研究）。OR-Tools 內集合了多個求解器，專門用於解決世界上最棘手的車輛路線規劃、作業流程、整數和線性程式設計及限制程式設計。針對於排程問題，它能夠在短時間內就能給出強大的解決方案，因此有許多研究人員會以 OR-Tools 作為基準，判斷自己提出的演算法之有效性。

求解排程問題會使用到 OR-Tools 裡的 CP-SAT 求解器（Constraint Programming with SAT），用於解決約束編程問題。約束編程是一種建模和求解問題的技術，其中問題的限制條件以約束的形式給出。CP-SAT 求解器利用了 Boolean，滿足 SAT 求解器與約束編程的結合，提供了一種高效靈活的方法來解決多種問題。

CP-SAT 求解器主要的優勢是「高效求解」，利用先進的求解技術和優化演算法，可以在合理的時間內找到高質量的解。它具有強大的搜索和剪枝能力，能夠處理大規模和複雜的問題。

CP-SAT 求解器在多個領域和應用中得到了廣泛的應用，例如排程問題、路線規劃、資源分配、排他性約束等。它已經被證明在工業界和學術界中具有重要的價值，並為使用者提供了一個有效解決問題的工具。

因此在本研究中，我們將會使用到 OR-Tools 來驗證我們提出方法的有效性。

第三章 研究方法設計與實作

根據第 1.2 節研究目的所述，本研究的目的是開發一種基於深度強化學習的動態調度框架，以最小化 DFJSP 中的 Makespan。再根據 2.2.1 小節所述，我們以 L2D 方法做為基礎，將其擴展為能處理具有動態機台數量的 FJSP，以實現本論文要達成的目標。

本章會提出新的狀態模型、特徵值、動作模型和獎勵函數，並對 L2D 方法中的步驟進行修改。我們將以小節分別論述這些新提出的方法。

3.1 狀態模型

根據 L2D 架構，狀態一開始會輸入進 GIN。狀態的資料結構為圖形，因此需要給定圖形的鄰接矩陣（adjacency matrix）和每一個節點的特徵值（features），鄰接矩陣用於表示圖形的結構以及節點和其鄰居節點之間有向邊的權重。由於現有的圖形表示無法詮釋 FJSP，因此我們要從圖形結構下手，向圖形添加更多的節點和有向邊。關於此細節將於 3.1.1 小節介紹。

JSSP 的每種機器只有一台，現有 L2D 方法的兩組特徵即可區分出不同任務的差異性。但是當改成 FJSP 時，由於相同種類的機器在處理相同任務時，所需要花費的時間相同，因此現有特徵值無法區分出同一個任務但是由不同機器處理的差異性，因此我們需要提出更多的特徵，以更好區分它們之間的差別，關於此細節將於 3.1.2 小節介紹。

3.1.1 處理多台機器

我們提出了 2 種新的圖形表示方法，首先假設 FJSP 由一組工作 $J = \{J_1, J_2, \dots, J_n\}$ 和一組機器類型 $M = \{M_1, M_2, \dots, M_m\}$ 所組成，每種機器類型 M_k 由一組相同類型的機器組成 $M_k = \{M_{k1}, M_{k2}, \dots, M_{kl_k}\}$，其中 l_k 表示在機器類別 M_k 上擁有的機器數量。每一個工作 J_i 都包含多個任務集合 O_{ij}，O_{ij} 依照對應機器類型的機器數量有多個平行任務 $O_{ijq} \in O_{ij}$，其中 $1 \le j \le m_i$，$1 \le q \le l_k$。O_{ijq} 處理時間表示為 $p_{ij} \in \mathbb{N}$，即 $\{O_{ij1}, O_{ij1}, \dots, O_{ijl_k}\}$ 中的每一個任務處理時間都是 p_{ij}。

- 第一種圖形表示方法：

定義初始狀態 s_0 分離圖 $G=(O,C)$，C 為合取的集合，即任務之間優先約束的有向邊集合。當 $O_{i(j+1)q}$ 存在時，$O_{ijq} \rightarrow O_{i(j+1)q}$，$1 \leq q \leq l_k$，此為節點之間的優先約束。節點$O_{ijq}$ 只能由一台對應的機器 M_{kr} 來處理。當發生增減機器事件時，假設要刪除機器 M_{kr}，那麼可以直接從分離圖 G 中移除由 M_{kr} 處理的節點集合$\{O_{1jq}, O_{2jq} \ldots, O_{ijq}\}$，並且與$\{O_{1jq}, O_{2jq} \ldots, O_{ijq}\}$有關的優先約束也會被移除。若當添加一台新機器$M_{k(r+1)}$，則增加由 M_{kr} 處理節點 $\{O_{1j(q+1)}, O_{2j(q+1)}, \ldots, O_{ij(q+1)}\}$，並添加對應的優先約束。

如圖 3-1 所示，一個 2×3 的 FJSP 實例，假設 M_1 有一台機器$\{M_{11}\}$，M_2 有兩台機器$\{M_{21}, M_{22}\}$，M_3 有三台機器$\{M_{31}, M_{32}, M_{33}\}$，那麼 s_0 可以表示成圖 3-1 的左圖形。在原本 JSSP 的分離圖上對每一個工作的任務添加平行節點，表示相同任務但是由不同機器來處理，例如 $O_{131}, O_{132}, O_{133}$ 分別代表 O_{13} 由 M_{31}, M_{32}, M_{33} 進行處理，這些平行節點視同一個任務，只要其中一個節點被處理，其他平行節點將不再可選。不同任務之間的優先約束依然存在，因此添加平行節點後，節點的優先約束將會變成全連接，如 O_{121}, O_{122} 與$O_{131}, O_{132}, O_{133}$ 的全連接。

Suppose jobs $J=\{J_1, J_2\}$ and machines $M=\{M_1, M_2, M_3\}$

- $M_1=\{M_{11}\}$; $M_2=\{M_{21}, M_{22}\}$; $M_3=\{M_{31}, M_{32}, M_{33}\}$

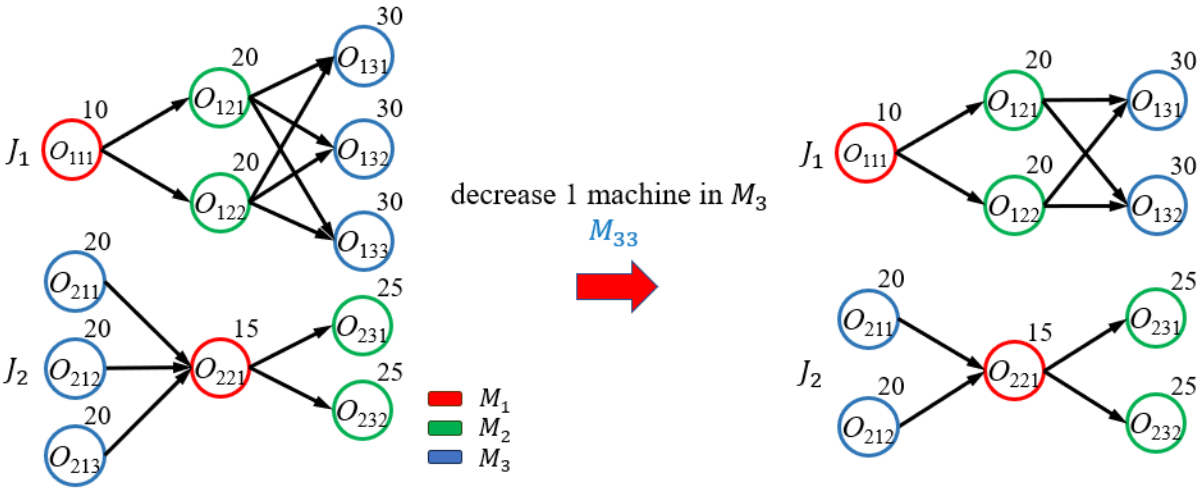

圖 3-1 FJSP 的分離圖表示以及減少一台機器後的表示

析取集合與 JSSP 的做法相同，將相同機器處理任務的順序添加有向邊，並放入析取集合當中。

當發生增減機器事件時，如圖 3-1 所示，減少機器類型 M_3 其中一台機器，這時會隨機決定要移除的機器，以機器 M_{33} 為例，移除機器 M_{33} 後只要將所有由 M_{33} 處理的節點進行移除即可，如圖 3-1 右圖形所示。這種圖形表示方法很清楚地將一個任務和一台機器的組合映射到一個節點上，並且只要沒有改變機器數量，合取集合就不會變化。缺點是圖形在較多的機器下會變得龐大。

- 第二種圖形表示方法：

該表示方法從第一種改變而來，目的是為了減少圖形複雜度。保留當前狀態 s_t 下的當前動作空間 A_t 的節點，已經選擇過的節點之其它平行節點都將刪除，因為這些節點也將不會再被代理選擇，屬於無用節點。在 A_t 中所有節點優先約束以後的節點，僅保留每一個工作的相同任務不同機器中最小「任務實際開始時間」的節點。關於實際開始時間最小的計算方式，請見第 3.1.2 小節。我們以圖 3-2 為例：

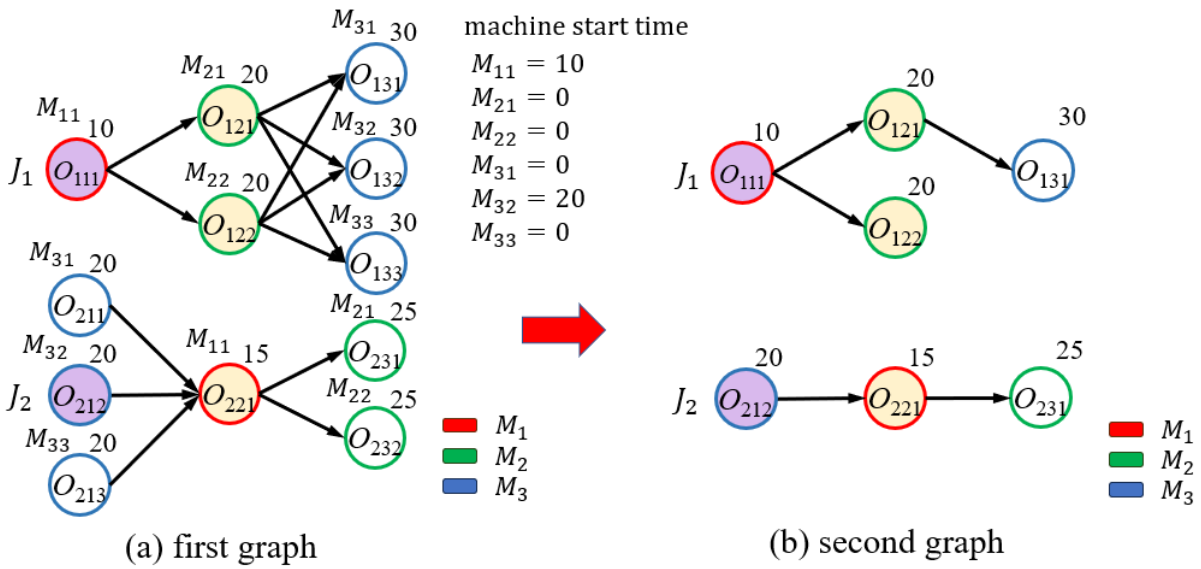

圖 3-2 從 FJSP 第一種的分離圖表示(a)轉成第二種分離圖表示(b)

左圖(a)為第一種圖形表示，右圖(b)為第二種表示方式。假設我已經選擇過 O_{111}, O_{212} 節點，那麼 O_{211}, O_{213} 則為無用節點，當前狀態下的動作空間 $A_t = \{O_{121}, O_{122}, O_{221}\}$，$\{O_{131}, O_{132}, O_{133}, O_{231}, O_{232}\}$ 是 A_t 中所有節點優先約束以後的節點，根據每一個工作的相同任務不同機器中最小「任務實際開始時間」的節點，我們僅保留 O_{131}, O_{231} 節點，若每一個工作相同任務不同機器的平行節點中，具有相同任務實際開始時間的節點有多個，則以機器編號排序為主，挑選第一個對應節點，在候選的可行任務中（A_t），O_{121}, O_{122} 為相同任務不同機器的節點，將只保留最小「任務實際開始時間」的節點與以後節點的優先約束。這種圖形表示方法，僅保留 A_t 中的節點，其他任務的節點將只顯示一個節點，能有效縮減圖形的規模和複雜度。

3.1.2 特徵值

L2D 原本的特徵值只有兩組：1. 任務完成時間的下限。2. 任務是否已被排程。這兩組特徵雖然能區分不同任務之間的狀態，但是無法區分相同任務不同機器的狀態。因此我們添加以下新特徵來區分不同機器狀態：

- 任務實際開始時間

我們定義任務完成時間的下限為 $C_{LB}(O_{ijq}, s_t) = r_{ij} + p_{ij}$，若 $a_t \in O_{ij}$，則 r_{ij} 等同任務開始時間 S_{ij}，否則 r_{ij} 等同於 $O_{i(j-1)}$ 的結束時間 $C_{i(j-1)}$。p_{ij} 為任務處理時間。任務完成時間的下限只有在動作所選擇的任務，其優先約束以後的節點才會被更新，並非任務實際完成時間。因此我們的任務實際開始時間的計算公式為(3.1)式：

$$S_{ijq} = \max\left(C_{LB} - p_{ij},\ \mathrm{S}(M_{kr})\right) \tag{3.1}$$

$\mathrm{S}(M_{kr})$ 為機器 M_{kr} 的開始時間，由於任務開始時間會受到前一個任務的結束時間和機器的開始時間所影響，該任務必須等到前一個任務結束後並且機器有空閒時才能開始處理，因此會取兩者最大值作為任務實際開始時間。為何不選擇「任務實際減少時間」或是單純使用「機器開始時間」作為新特徵？我們將於第四章的實驗中給出答案。任務實際開始時間在 FJSP 中的計算方式以圖 3-3 為例：

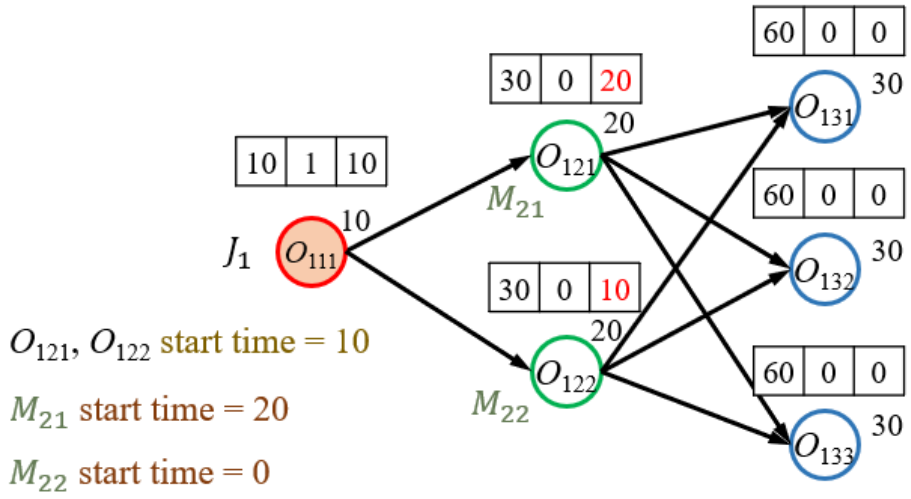

圖 3-3 任務實際開始時間在 FJSP 中的表示

每一個節點上方方框內的三個數值為特徵值，三個數值最右邊為新添加的第三特徵「任務實際開始時間」，假設當前狀態下的動作空間為 $A_0 = \{O_{121}, O_{122}\}$，$M_{21}$ 由於有處理其他任務，所以當其他任務做完時結束時間為 20，機器處理 O_{121} 開始時間為 20，M_{22} 尚未處理過其他任務，所以 M_{22} 處理 O_{122} 的開始時間為 0。若沒有加上第三特徵，則 O_{121}, O_{122} 的特徵值相同，無法區分兩者差別，根據前述任務實際開始時間計算公式，O_{121} 的實際開始時間 $S_{121} = \max(\text{任務完成時間的下限} - \text{任務處理時間}, \text{機器 } M_{21} \text{的開始時間}) = \max(30 - 20,\ 20) = 20$，而 O_{122} 的實際開始時間 $S_{122} = \max(30 - 20, 0) = 10$，因此根據任務實際開始時間能區分由不同機器處理相同任務，讓代理能夠好學習特徵。

我們在實驗中還添加了額外的特徵：

- 任務處理時間總和

該特徵計算方式類似於 PDR 中 MWKR 方法的 WKR 計算方式，該數值等於該任務優先約束以後的所有任務處理時間總合，多平行節點的任務以其中一個計算，如圖 3-3 的 O_{111} 任務處理時間總和，會等於 $\{O_{121}, O_{122}\}$ 其中一個處理時間再加上 $\{O_{131}, O_{132}, O_{133}\}$ 其中一個處理時間，即 20+30=50。由於在我們的實驗中證明 MWKR 在處理 DFJSP

時擁有最好的效能，因此我們將此指標用於做為第 4 特徵，並在實驗中確認它的有效性。

3.2 動作模型

定義動作 $a_t \in A_t$，a_t 為決策步驟 t 下的一個可行任務。動作空間 A_t 會根據當前狀態 s_t 而有所變化。與 L2D 的方法相同，動作空間依然是當前狀態的可行的任務，可行的任務會受到優先約束的影響，如果前面任務還沒被選擇過，則後面任務就是不可行的。

根據上一節兩種圖形表示方法，會有兩種狀態轉移模型。圖 3-4 為第一種圖形表示方法的狀態轉移範例，假設在初始狀態 s_0 下機器 M_{31} 的開始時間為 7，M_{33}的開始時間為 3，其餘皆為 0，初始狀態 s_0 的動作空間 $A_0 = \{O_{111}, O_{211}, O_{212}, O_{213}\}$，當選擇 O_{211} 做為動作時，動作空間轉變為 $A_1 = \{O_{111}, O_{221}\}$，並更新 O_{221} 與其平行節點和其優先約束以後所有節點的第一特徵「任務完成時間的下限」，更新 O_{221} 的第二特徵「任務是否已被排程」為 1，更新所有由 M_{31} 所處理的任務節點 O_{131} 和 O_{211} 的第三特徵「任務實際開始時間」，更新後的狀態如圖 3-4 所示。

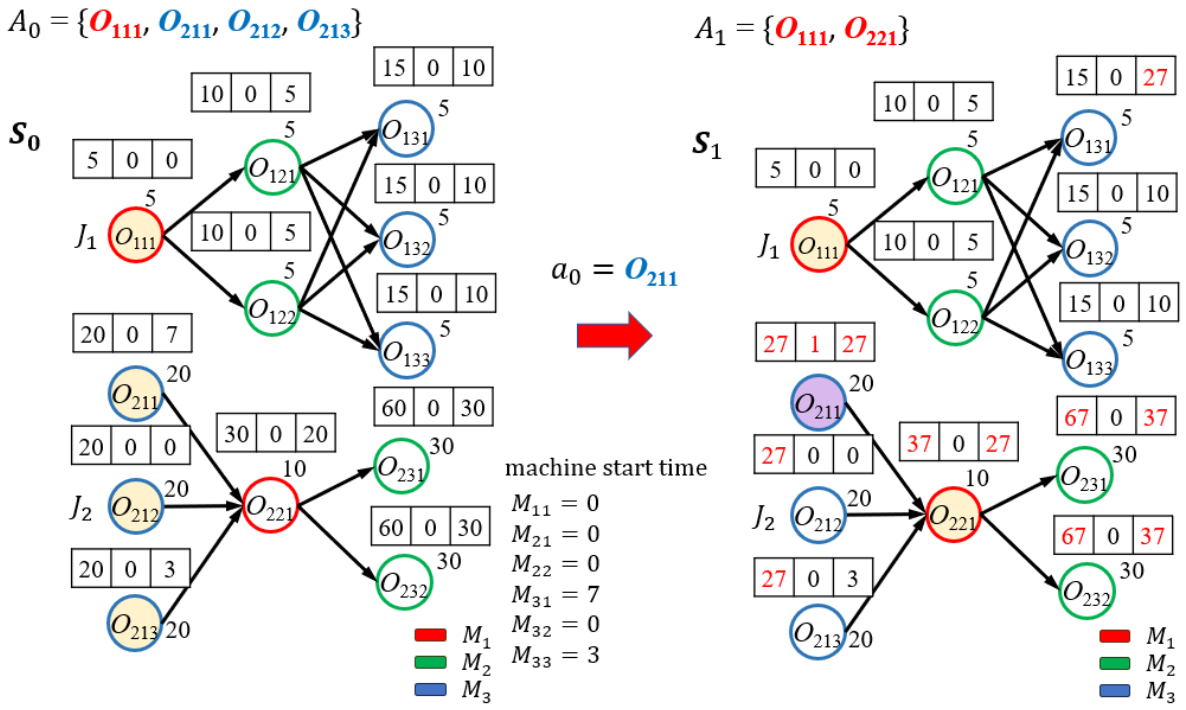

圖 3-4 第一種圖形表示方法的狀態轉移

圖 3-5 為第二種圖形表示方法的狀態轉移範例，當選擇 O_{211} 做為動作時，從第一種圖形表示方法轉換而來，保留當前狀態 s_t 下的當前動作空間 A_t 的節點，已經選擇過的節點之其它平行節點 O_{212} 和 O_{213} 都將刪除，因為這些節點也將不會再被代理選擇，屬於無用節點。在 A_1 中所有節點優先約束以後的節點，僅保留每一個工作的相同任務不同機器中最小「任務實際開始時間」的節點，即保留 O_{132}。更新後的狀態如圖 3-5 所示。

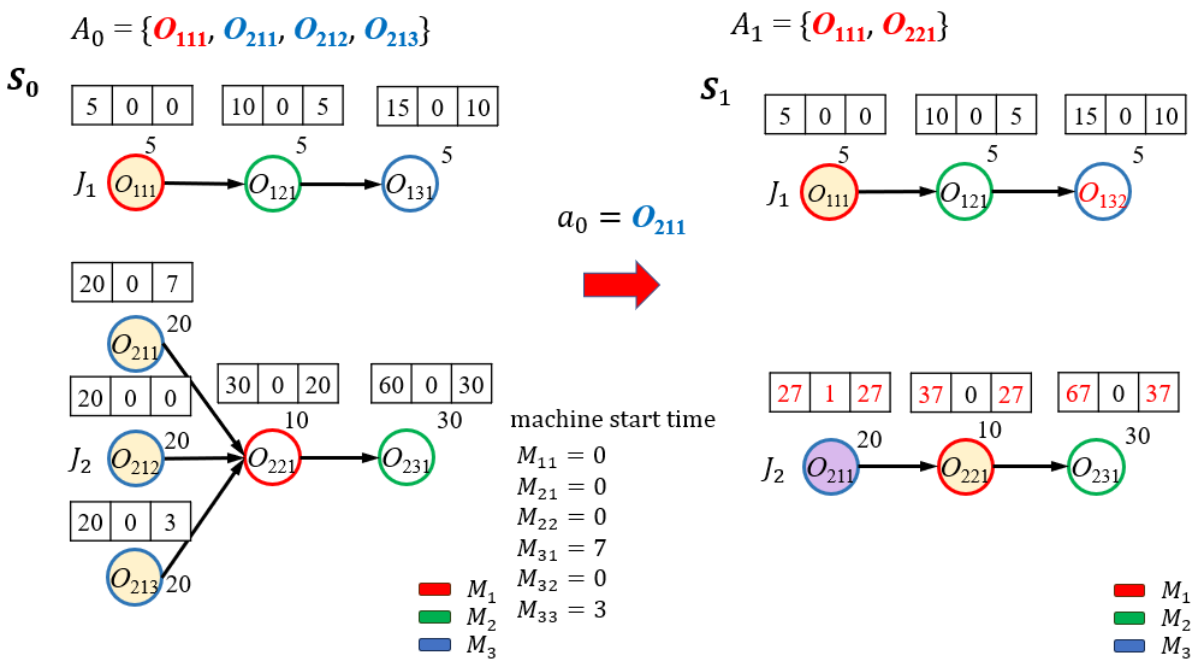

圖 3-5 第二種圖形表示方法的狀態轉移

3.3 圖形池化

在 GIN 的最後階段，會將 Projection 輸出的特徵向量經過圖形池化(Graph Pooling)，將整個圖形壓縮成一個節點，使該節點具有整個圖形的資訊。由於第一種圖形表示方法會包含沒有在排程中出現的同任務不同機器的節點，即無用節點。若將這些無用節點作為圖形的一部份進行 Graph Pooling，將會使其參雜一些無用資訊，進而影響狀態嵌入的效率。因此我們會針對第一種圖形表示方法移除無用節點後再進行 Graph Pooling。

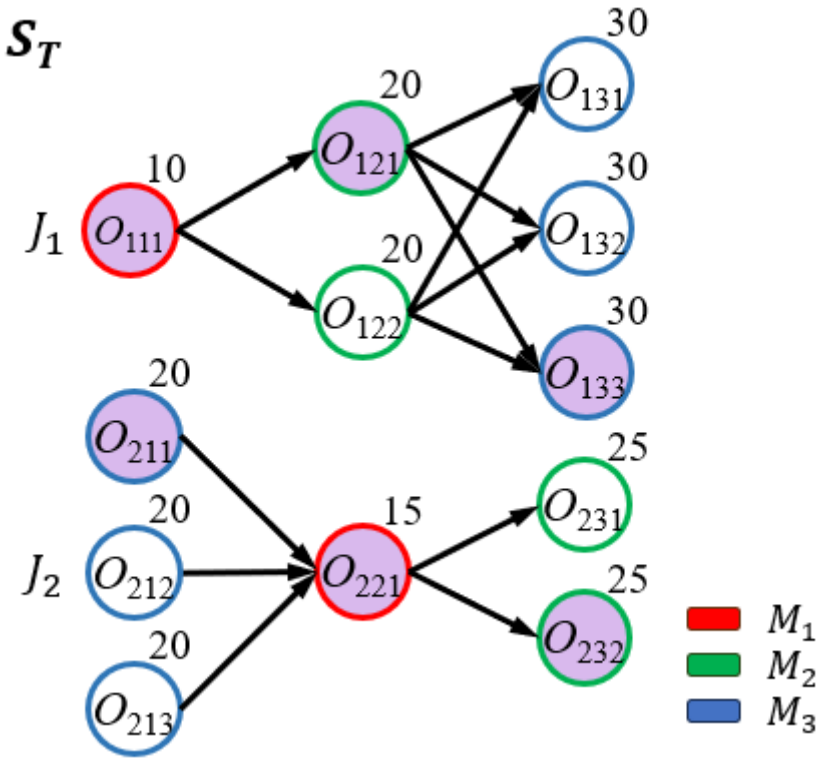

圖 3-6 第一種圖形表示方法的中止狀態

圖 3-6 是一個 FJSP 的中止狀態，已經確定好兩個工作 J_1 和 J_2 中所有任務要分配給哪一些機器進行處理，被選擇的節點有 $\{O_{111}, O_{121}, O_{133}, O_{211}, O_{221}, O_{232}\}$，其他節點都是屬於無用節點。Graph Pooling 的做法是平均池化，在圖 3-6 中共有 12 節點，因此每個節點特徵值都會乘上 $\frac{1}{12}$，之後再將所有節點特徵值進行加總，成為一個包含整個圖形資訊的特徵值。我們的做法是將無用節點共 6 個節點的特徵乘上 0，其餘 6 個節點乘上 $\frac{1}{6}$，再加總所有節點特徵值。

3.4 獎勵函數

原有的 L2D 獎勵函數定義為$R(a_t, s_t) = H(s_t) - H(s_{t+1})$，其中 $H(s_t)$ 是 Makespan 下限，即 $\max_{i,j,q}\{C_{LB}(O_{ijq},\ s_t)\}$。除了原有的獎勵函數之外，我們還採用了其他計算方式。以下列出本論文實驗中採用的計算目標：

- 剩餘任務處理時間：

定義$R(a_t, s_t) = H(s_t) - H(s_{t+1})$，$H(s_t) = \max_i\{W(J_i,\ s_t)\}$，$W$ 會計算工作 J_i 在狀態 s_t 下還未處理的總任務處理時間。

- 任務等待機器開始處理的時間：

定義$R(a_t, s_t) = \mathrm{S}(M_{kr}) - C_{LB} + p_{ij}$，$\mathrm{S}(M_{kr})$ 為機器M_{kr}的開始時間，C_{LB} 為該任務的完成時間下限，p_{ij} 為任務的處理時間。此獎勵目標是讓代理盡可能選擇不需等待的機器進行任務處理，使該工作的前一個任務完成後，能接續處理下一個任務。

- 機器上下限比率：

在 DFJSP 中，每種機器類型的機器數量會隨時間變化，但有設定其機器數量的上限和下限，因此根據當前機器數量在機器上下限的比率來計算獎勵。定義每種類型機器的數量上限為 U_{M_i}，每種類型機器的數量下限為 L_{M_i}，每種類型機器的當前數量為$C_{M_i} = \{L_{M_i}, L_{M_i}+1, \dots, U_{M_i}\} \in \mathbb{N}$。獎勵函數為 $-\frac{U_{M_i}-C_{M_i}}{U_{M_i}-L_{M_i}}$，當 U_{M_i} 等同於 L_{M_i}，則獎勵值為一個常數 -0.5。目的是為了讓代理優先選擇接近機器數量上限的機器，數量多的機器較能找到閒置的機器用於分配任務。

Lu Duan 等人[10] 採用一種帶有 deterministic greedy rollout 的 REINFORCE 演算法來訓練策略。因此根據其想法，我們新增了一個 Baseline 策略用於計算獎勵。計算獎勵流程如圖 3-7 所示。

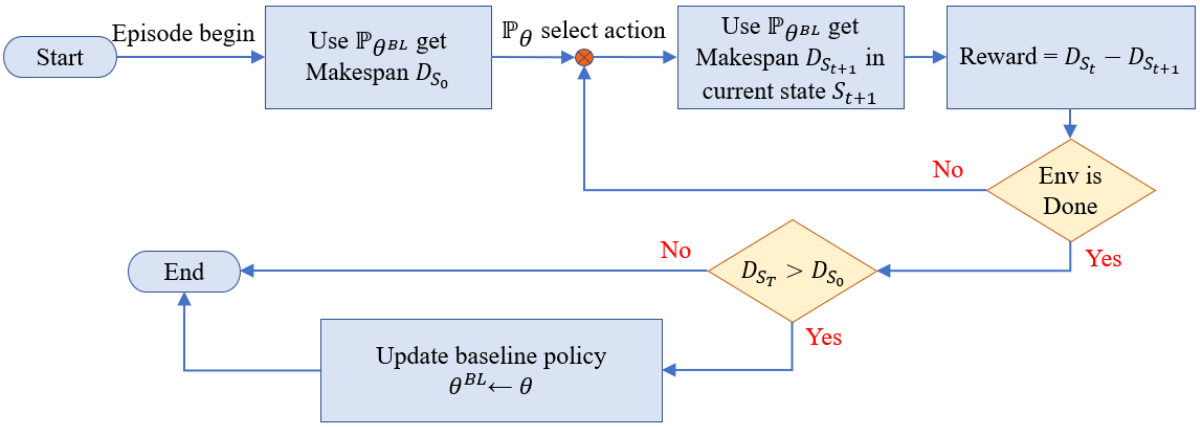

圖 3-7 使用 Baseline 策略計算獎勵的流程

定義 P_θ 為 PPO 策略，$P_{\theta^{BL}}$ 為 Baseline 策略，一開始的 Baseline 策略的參數為隨機亂數。該流程整體的做法是，Baseline 策略用於計算兩個狀態前後的 Makespan，若 Makespan 有變好，則給予正獎勵，否則為負獎勵。這兩種策略會互相扶持，當 PPO 的策略變好時，Baseline 策略也會變好，進而使計算出的獎勵更有參考價值。

在一個 episode 開始時，先使用 $P_{\theta^{BL}}$ 求解出訓練樣本（DFJSP）的 Makespan，定義為D_{S_0}，接著在每一個狀態 s_{t+1} 時，以 $P_{\theta^{BL}}$ 求解當前剩餘未分配排程的 DFJSP 獲得新 Makespan $D_{S_{t+1}}$，該排程前半部分的動作為 P_θ 所選擇，後半部分為 $P_{\theta^{BL}}$ 選擇。獎勵值 $R(a_t, s_t) = D_{S_t} - D_{S_{t+1}}$。隨著 t 的推移，由 Makespan $D_{S_{t+1}}$ 由 $P_{\theta^{BL}}$ 選擇動作越少，當狀態來到 s_T 時，D_{S_T} 全由 P_θ 解出。最後比較 D_{S_T} 和 D_{S_0}，若 $D_{S_T} > D_{S_0}$，即為由 P_θ 解出的 Makespan 會比由 $P_{\theta^{BL}}$ 解出的 Makespan 還要好，因此將 PPO 策略的參數 θ 替換掉 Baseline 策略 θ^{BL}，進行下一個 episode 的迭代。

這種作法重點是 PPO 策略選擇的動作會帶來較好的 Makespan 則給予正獎勵，否則給予負獎勵，只要當 PPO 策略變好時，就會改善 Baseline 策略，進而相輔相成，實現增強學習。

3.5 PPO 策略更新

首先說明 L2D 內的神經網路的輸入和輸出尺寸，如圖 3-8 所示。n 代表分離圖在狀態 s_t 下的節點數量，f 為特徵數量，c 為在狀態 s_t 下的動作空間 A_t 內的節點數量（候選節點），由於是要處理動態問題，因此 n 和 c 會隨著 s_t 而變化。若特徵值定為 3 組，則 $f = 3$。n 和 c 做為 batch size 輸入進 MLP，因此不會出現尺寸不同而無法輸入。要用 PPO 來預測動作時，使 Policy 網路輸出尺寸為 $(c, 1)$，代表每一個候選節點的分數，最後以軸為 1 的方式將每一個候選節點進行 Softmax，以獲得動作機率。

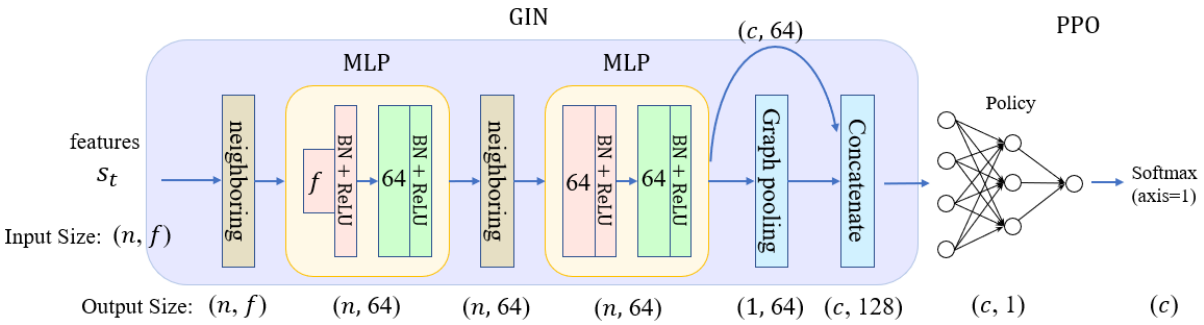

圖 3-8 L2D 架構的神經網路之輸入與輸出尺寸

不過在 PPO Update 時就會出現問題，這是因為 PPO 會將一個 episode 收集到的狀態和動作進行堆疊，組成一個較大的向量，再輸入策略網路來獲得 logprobs, state values, dist entropy 這三組向量，以便後續更新神經網路的參數。PPO Update 的部分虛擬碼如 **Algorithm 1** 所示。若 c 在整個 episode 中都沒有變動，則沒問題；若 c 在整個 episode 中有變動，則會因為 s_t 之間尺寸不同而無法堆疊。

Algorithm 1: Evaluating old actions and values for PPO update

1: Optimize policy for K epochs: evaluating old actions and values.

2: logprobs, state values, dist entropy ← P_θ(stack($[s_0, s_1, \ldots, s_T]$), stack($[a_0, a_1, \ldots, a_T]$))

有一種方式是確定 c 的上限，將 s_t 轉成相同尺寸，以數值 0 進行填充，這樣一來即可解決問題。不過此做法會使 L2D 的表現變差，因此我們將 **Algorithm 1** 改成 **Algorithm 2** 來處理尺寸不同產生的問題。

Algorithm 2: Evaluating old actions and values for new PPO update

1: Optimize policy for K epochs: evaluating old actions and values.
2: **for** $i = 0, 1, \ldots, T$ **do**
3: $\pi_i, v_i, e_i \leftarrow \mathrm{P}_\theta(s_i, a_i)$
4: **end for**
5: logprobs, state values, dist entropy ← stack($[\pi_i, \ldots]$), stack($[v_i, \ldots]$), stack($[e_i, \ldots]$)

Algorithm 2 的做法是讓策略網路在更新時的輸入 s_i 和 a_i，與策略網路在做動作預測時的輸入 s_i 和 a_i 是相同的，將輸出得到的 π_i, v_i, e_i 三個向量在全部的狀態和動作輸入完後再堆疊起來。經過實驗證明，在無動態事件的 JSSP 上用此方法訓練策略，會比原有 PPO Update 的方法還要帶來更好的 Makespan，因此該作法是可行的。

第四章 實驗結果與討論

在本章中，我們會介紹實驗環境與所使用的資料集，接著會定義實驗中所使用的動態機台數量增減事件發生機率公式。隨後使用第三章中提到的各種方法進行實驗。

4.1 實驗環境設置

用於實驗的電腦有三台，電腦硬體設備如表 4-1 所示。本實驗所使用的程式語言為 Python，深度強化學習框架為 PyTorch，相關套件版本如表 4-2 所示。我們所使用的 GIN 和 PPO 神經網路的超參數設置如表 4-3 和表 4-4 所示。

表 4-1 實驗用電腦硬體設備

處理器	Intel Core i7-9700 3.00GHz x 8	Intel Core i7-7700 3.60GHz x 8	Intel Core i7-4770 3.40GHz x 8
記憶體	16 GB	40 GB	16G
作業系統	Windows 10	Ubuntu 18.04	Ubuntu 18.04

表 4-2 Python 相關套件版本

函式庫套件名稱	版本
Python	3.10
torch	1.13.1
ortools	9.6.2534

表 4-3 GIN 超參數設置

Aggregate Type	Sum
Graph Pooling Type	Average
Activation Function	Batch Normalization + ReLU
Input Shape	(Batch size, 3)
Output Shape	(Batch size, 128)
Batch Size	Number of Node in Input Graph

表 4-4 PPO 超參數設置

Num of Envs	4
Activation Function	Tanh
Output Activation Function	Softmax
Loss Function	Mean Squared Error
Optimizer	Adam
Discount Factor	1.0
Input Shape	(Batch size, 128)
Hidden Shape	(Batch size, 64)
Output Shape	(Batch size, 1)
Batch Size	Number of Node in Input Graph

4.2 資料集

關於實驗中所使用的資料集都是經由亂數產生。本節會說明 FJSP 的初始機台數量公式，若在實驗中不指定任何其他條件的話，都以此處的設置為主。隨後會說明動態機台數量增減事件發生機率，以及兩個動態事件到來的間隔。

定義工作數量 $= x$；機器類型數量 $= y$；每一個工作內的任務數量 $= y$；每一個任務要處理的時間$\in [d_{min}, d_{max}]$，實驗中 d_{min} 設為 1，d_{max} 設為 99；FJSP 實例大小表示為 $|x| \times |y|$。根據上述變數，我們在初始化一個 FJSP 的機器數量時，每一機器類型的初始數量為 $M_i = n \pm 20\%$，其中$M_i \in \mathbb{N}$ ，$i = \{1, 2, \dots, y\}$，所有機器類型初始機器數量的平均數 $\frac{1}{y}\sum_{i=1}^{y} M_i$ 會等同於 n，n 在(4.1)式中給出。

$$n = \log_2\left(\frac{x}{y}\right) + 1 \tag{4.1}$$

(4.1)式的用意是為了讓工作數量與機器類型數量的比值變大時，每種機器類型的初始數量也會跟著變大，我們還有加上 $\log_2$ 的目的是讓機器初始數量的增長不成正比，因為機器數量不可能與工作數量一樣多。定義每種機器的數量上限為 $U_{M_i} = M_i$，每種機器的數量下限為 $L_{M_i} = 1$，每種機器的當前數量為$C_{M_i} = \{L_{M_i}, L_{M_i} + 1, \dots, U_{M_i}\} \in \mathbb{N}$。

對於每種機器類型，在 t 時間內，平均發生 λ 次機器增加或減少事件，符合 Poisson 分布。其中 t 在(4.2)式中給出，λ 在(4.3)式中給出。

$$t = \frac{d_{min} + d_{max}}{2} xy \tag{4.2}$$

$$n = \frac{xy}{n} \tag{4.3}$$

(4.2)式讓 t 等同於所有任務處理時間的總和，意即排程最差的情況下可能會達到的數值，(4.3)式讓 λ 在機器數量越多時，以相同比例增加發生次數。發生事件時，根據機率選擇機器增加事件或是機器減少事件，增加機器事件的機率為 $\frac{U_{M_i} - C_{M_i}}{U_{M_i} - L_{M_i}}$，減少機器事件的機率為 $1 - \frac{U_{M_i} - C_{M_i}}{U_{M_i} - L_{M_i}}$ ，兩種事件的機率總和為 100% 且不會同時發生。機率呈線性增長，機器增加量和減少量為一個隨機整數，但不會使機器超過其上限 U_{M_i} 和下限 L_{M_i}。

在實作中，由於機器初始數量是一個包含小數點的範圍，我們需要將隨機亂數產生的數值轉為整數，因此我們會給予機器初始數量可能的數值一個比重，讓隨機亂數依照比重而傾向產生對應數值，使其平均數等於 n。舉例三種不同實例大小的問題，其參數如表 4-5 所示，機器初始機器數量的比重如表 4-6 所示。

表 4-5 FJSP 的初始機器數量範圍和動態事件的平均發生時間

問題實例大小	15×5	150×5	15×15
$[d_{min}, d_{max}]$	[1, 99]	[1, 99]	[1, 99]
M_i 範圍	[2.067, 3.101]	[4.725, 7.088]	[0.8, 1.2]
n	2.5849	5.9068	1.0
事件平均發生時間$\frac{t}{\lambda}$	129	295	50

表 4-6 FJSP 的初始機器數量範圍比重

問題實例大小	15×5	150×5	15×15
$M_i = 0$的比重	0	0	0
$M_i = 1$的比重	0	0	1
$M_i = 2$的比重	0.933	0	0
$M_i = 3$的比重	1	0	0
$M_i = 4$的比重	0.101	0.274	0
$M_i = 5$的比重	0	1	0
$M_i = 6$的比重	0	1	0
$M_i = 7$的比重	0	1	0
$M_i = 8$的比重	0	0.088	0

在代理進行學習時，每一個 episode 都會隨機產生一個 DFJSP 實例用來學習，因此訓練集的數量會根據 episode 來決定。測試集的大小為 30 筆 DFJSP 實例，在本論文實驗中，每經過一定數量的 episode 後，就會將目前已學習的策略用來求解測試集，並評估 30 筆測試集實例的平均 Makespan，已了解目前學習的策略是否過擬合，或是否能有效的學習策略，解決 DFJSP 問題。

另一種實驗評估方式是，當模型已經收斂，無法再降低 Makespan 之後，將此模型策略用於評估 30 筆測試集實例的平均 Makespan，並比較其他方法對於這 30 筆測試集實例的平均 Makespan，藉此來了解不同方法之間的有效性。

4.3 研究問題

為了探討本論文擴展 L2D 後所提出的新方法是否能夠處理具有動態機台數量的 FJSP，且該新方法效能是否能夠優於傳統 4 種強大的 PDR，因此我們設計了以下四個研究問題：

RQ1. 我們提出的特徵是否比原始特徵更好地解決了 FJSP？

RQ2. 我們提出的獎勵函數是否比原始獎勵函數表現更好？

RQ3. 我們提出的方法是否比現有最強的 PDR 表現更好？

RQ4. 訓練後的模型是否可以解決不同大小的 DFJSP 實例？

4.4 實驗一：使用不同的狀態模型來處理 FJSP

為了驗證我們提出的特徵是否比原始特徵更好地解決了 FJSP，我們設計實驗一來回答 RQ1。首先我們要嘗試兩種圖形表示方法和特徵種類在靜態的 FJSP 表現效果，之後的實驗才能加入動態機台數量來更進一步實驗。

我們的實驗方法是強化學習代理在訓練中，每經過一個固定次數（次數=50）的 episode 後，將目前訓練的模型進行一次測試集的預測，並將所得到的 Makespan 進行平均，最後將 Makespan 以折線圖的方式呈現出來。如此可以從中得知模型的效能，以及訓練是否出現過度擬合（overfitting）的情況。

圖 4-1 是使用 FJSP 實例大小為 6×6，且每種機器類型固定為 2 台機器的靜態問題，此實驗主要目標是驗證我們提出的特徵是否能有效表達相同任務但不同機器的動作，並在測試集的表現中是否良好？該實驗的狀態模型使用第一種圖形表示方法，圖形池化與獎勵函數使用原本 L2D 方法，PPO 策略更新一律使用我們提出的方法。

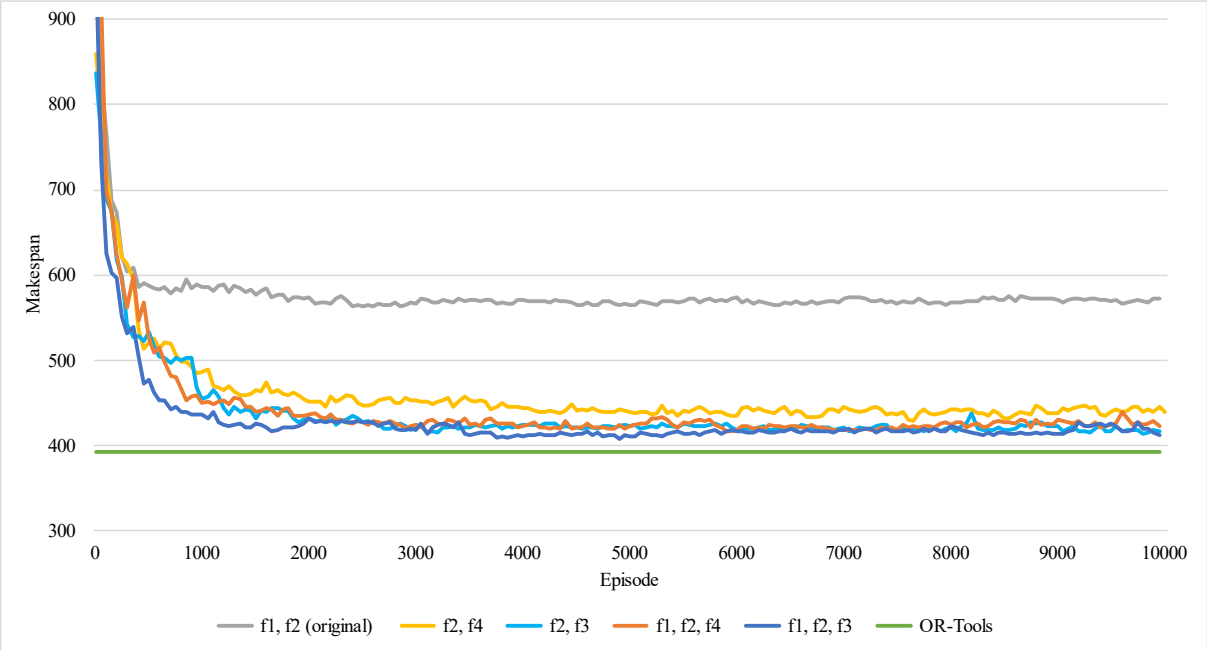

圖 4-1 不同特徵在 6×6 固定 2 台機器的 FJSP 下之學習曲線

圖 4-1 圖例中的 f1 為任務完成時間的下限（原有特徵），f2 為任務是否已被排程（原有特徵），f3 為任務實際開始時間（我們新提出特徵），f4 為任務實際開始時間+任務處理時間（即任務實際結束時間），OR-Tools 為實例的平均最佳解。從實驗結果可以得知，最好的組合為{ f1, f2, f3 }，擁有最快收斂速度，且後續的 episode 中相較其他的組合，擁有最低的 Makespan 成績。而{ f1, f2 }由於不使用新提出的特徵，無法區分出不同機器之間的差異，導致表現差。

表 4-7 為圖 4-1 在 10000 episode 下各種特徵組合的 Makespan。從表中可以得知最好的特徵組合為{ f1, f2, f3 }。

表 4-7 在 10000 episode 下各種特徵組合之 Makespan

	特徵組合					最佳解
	f1, f2	f2, f4	f2, f3	f1, f2, f4	f1, f2, f3	OR-Tools
Makespan	572	445.3	416.7	422.4	**412.3**	392.9

圖 4-2 是使用兩種不同的圖形表示方法在 FJSP 實例大小為 10×5，且每種機器類型固定為 2 台機器的靜態問題上之學習曲線。從圖中可以得知，在第一種圖形表示方法中，訓練時不會出現過度擬合的情況。第二種圖形表示雖然能夠快速學習到策略，不過隨著訓練的推移開始出現過度擬合的情況。我們認為這是第二種圖形表示移除了圖形中的部分節點，聚合鄰居時獲得的鄰居資訊較少，並且在進行圖形池化後，使嵌入特徵缺少了部份圖形的資訊。考慮到我們需要一個能處理不同實例大小和動態問題的情況，因此使用第一種圖形表示方法較為合適。若要快速學習到策略，使用第二種圖形表示方法也是可行的選擇。

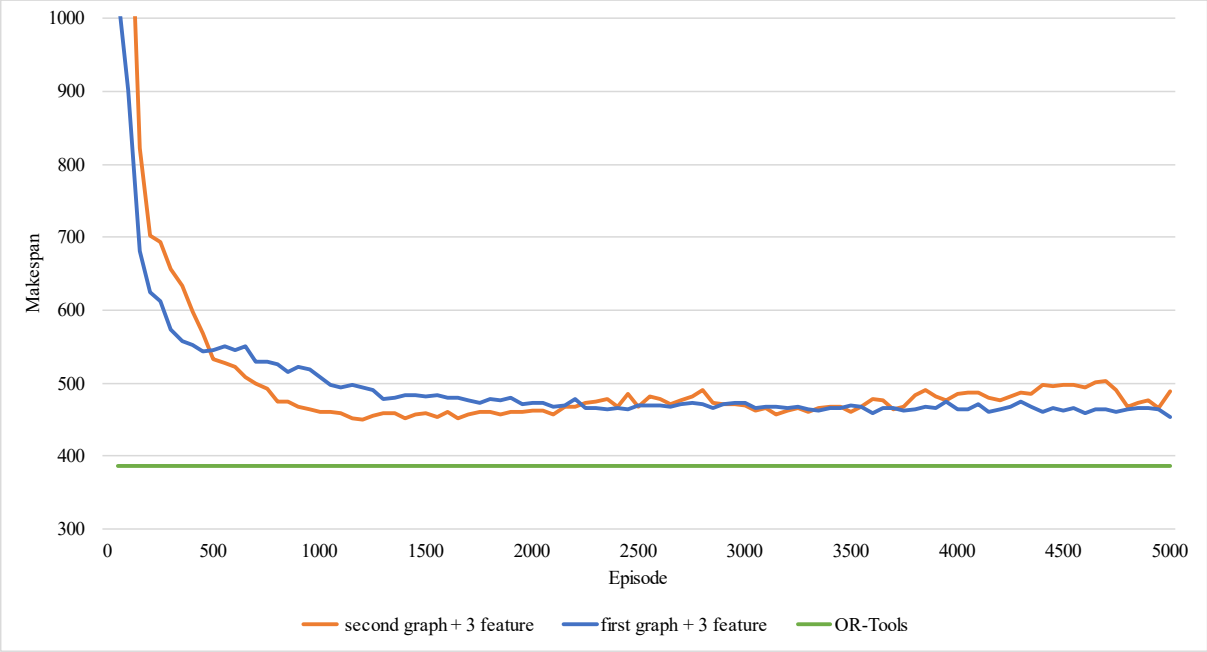

圖 4-2 不同圖形表示方法在 10×5 固定 2 台機器的 FJSP 下之學習曲線

圖 4-3 是使用不同的特徵數量在 FJSP 實例大小為 10×5，且每種機器類型固定為 2 台機器的靜態問題上之學習曲線。圖形表示方法採用第二種。我們以在表 4-7 得出的最好特徵組合作為基礎，額外添加了第四種特徵「任務處理時間總和」。該特徵於 3.1.2 小節有介紹。從圖中可以得知，添加第四種特徵後，表現有明顯的下降。我們認為這種特徵對於節點表示不是很好的指標，會干擾代理的學習。雖然在 PDR 中的 MWKR 以此作為指標取得不錯的成績，但在強化學習中表現不理想，因此不能將此特徵做為第四種特徵。

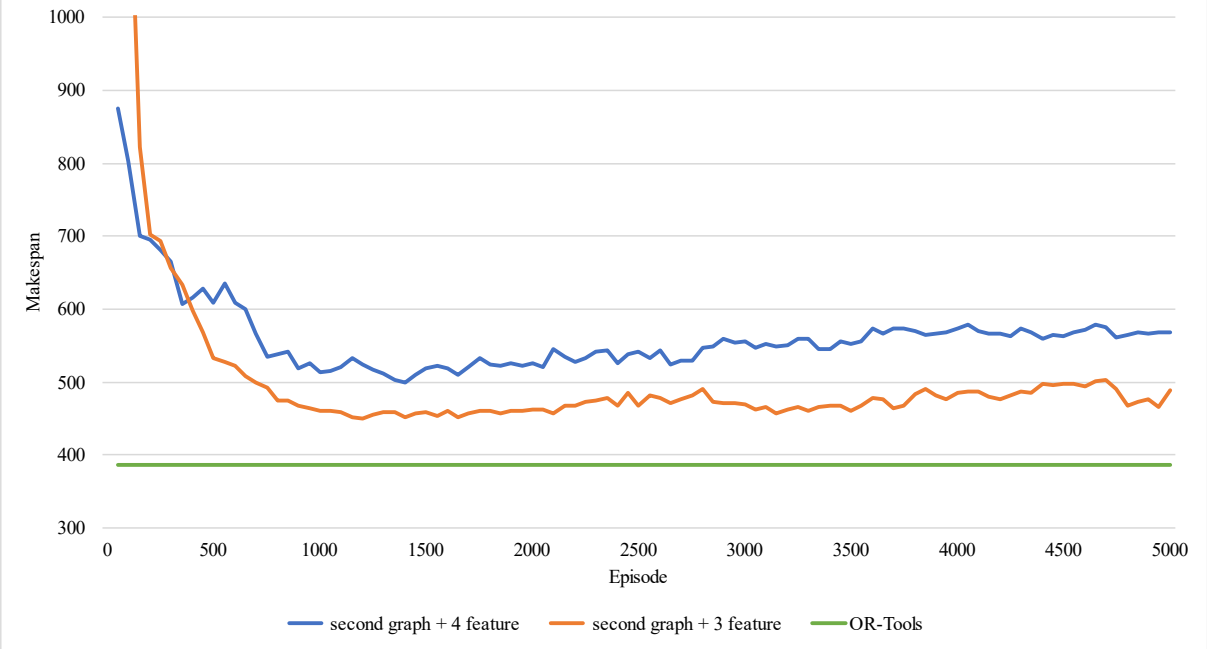

圖 4-3 不同特徵數量在 10×5 固定 2 台機器的 FJSP 下之學習曲線

根據上述的實驗結果，對於 RQ1 我們可以得出，我們提出的特徵有比原始特徵更好地解決了 FJSP。使用第一種圖形表示方法，並且在特徵組合為「任務完成時間的下限」、「任務是否已被排程」和「任務實際開始時間」的情況下，代理在訓練中表現穩定且不會出現過度擬合的情況，能夠得到最好的 Makespan。

4.5 實驗二：使用不同的獎勵函數來處理 DFJSP

為了驗證我們提出的獎勵函數是否比原始獎勵函數表現更好，我們設計實驗二來回答 RQ2。我們會在靜態和動態機台數量的 FJSP 上觀察不同獎勵函數下的代理，在學習曲線上的表現如何，並在較大的問題實例上評估不同獎勵函數下的代理之表現。

本實驗所使用的圖形表示方法為第一種圖形表示方法，特徵組合由實驗一表現出最好的組合進行配置。圖 4-5 是使用 L2D 原本獎勵函數和本文提出使用 Baseline 策略的獎勵函數，在 FJSP 實例大小為 10×5，且每種機器類型固定為 2 台機器的靜態問題上之學習曲線。從實驗結果可以看得出來使用 Baseline 策略優於原本獎勵函數。

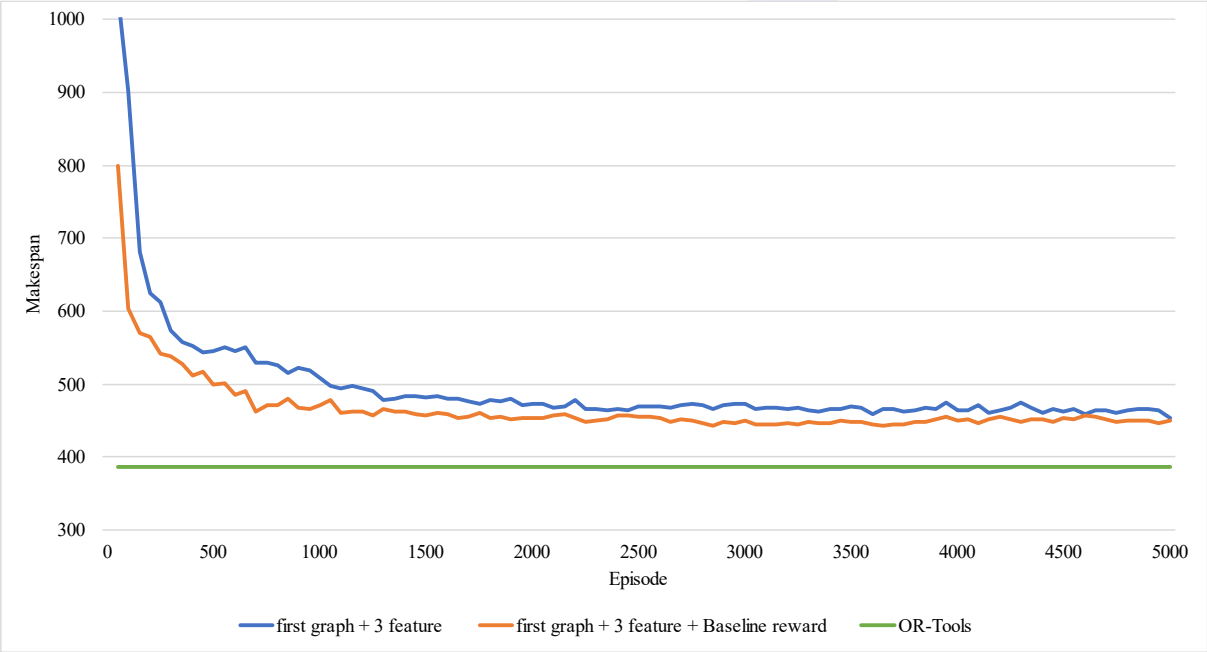

圖 4-4 不同獎勵函數在 10×5 固定 2 台機器的 FJSP 下之學習曲線

圖 4-5 是本文提出使用「剩餘任務處理時間」(work remaining)、「任務等待機器開始處理的時間」(waiting start)和「機器上下限比率」(machine rate)，以及使用 Baseline 策略的獎勵函數(baseline policy reward)與原本獎勵函數在動態機台數量的 6×3 實例大小的 DFJSP 下之學習曲線。

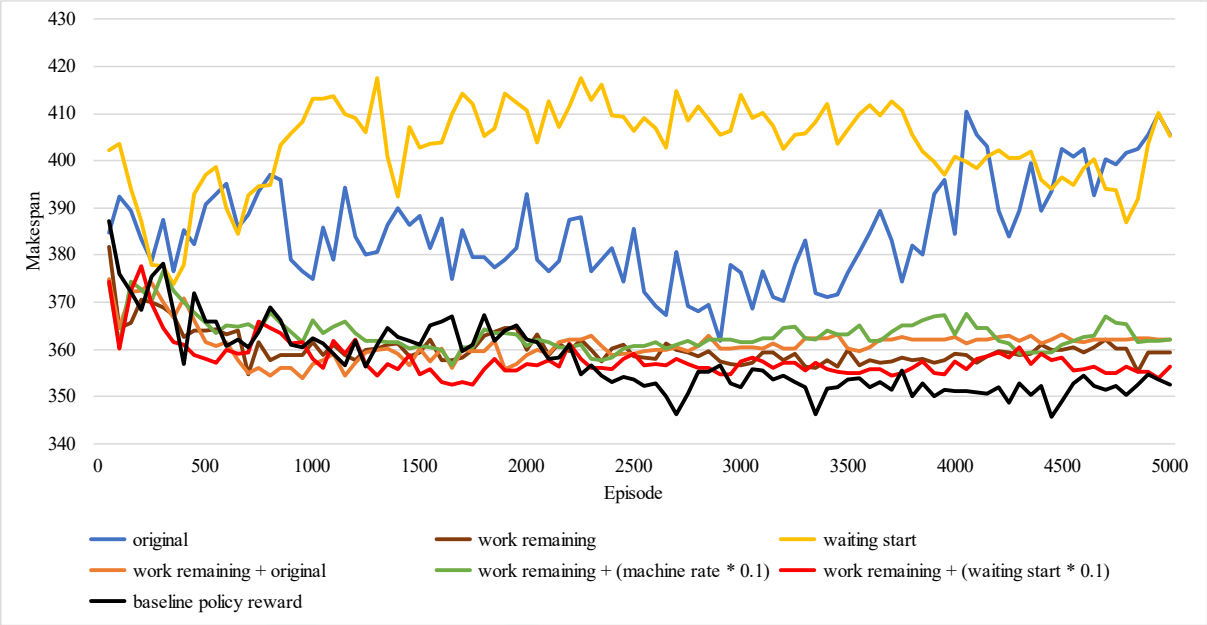

圖 4-5 不同獎勵函數在 6×3 的 DFJSP 下之學習曲線

從圖 4-5 可以得知，baseline policy reward 能夠獲得最好的 Makespan。原始獎勵函數在動態問題上的學習曲線並不穩定，若撇除 baseline policy reward，獎勵函數單獨使用 work remaining 就有良好表現，學習曲線相較於原始獎勵函數較為穩定，並且在使用 work remaining 加上 0.1 係數 waiting start 的複合獎勵函數中表現最佳。

表 4-8 是不同獎勵函數在 15×5 的 DFJSP 下的 Makespan。compound reward 為圖 4-5 表現最佳的複合獎勵函數。Baseline $P_{\theta^{BL}}$=PPO 是本論文提出的 Baseline 策略。另外我們添加了一個新獎勵函數 Baseline $P_{\theta^{BL}}$=MWKR 進行比較，該獎勵函數同樣使用 Baseline 策略獲取獎勵，只不過在初始狀態 s_0 的 Baseline 策略是使用 OR-Tools，在初始狀態 s_1 到 s_T 之間是使用 PDR 中的 MWKR 方法。在 episode 結束並沒有 Update baseline policy 的階段，該方法目的是試圖讓代理學習 MWKR 的策略，實驗其是否會超越我們其他提出的方法。

表 4-8 不同獎勵函數在 15×5 的 DFJSP 下之 Makespan

	DRL 獎勵函數				最佳解
	original	compound reward	Baseline $P_{\theta^{BL}}$=MWKR	Baseline $P_{\theta^{BL}}$=PPO	OR-Tools
Makespan	744.4	731.0	703.3	**697.7**	568.1

從表 4-8 中可以得知，使用 Baseline $P_{\theta^{BL}}$=MWKR 會比我們的複合獎勵函數還要好，但是 Baseline $P_{\theta^{BL}}$=PPO 是在這三種獎勵函數中表現最好的。

根據上述的實驗結果，對於 RQ2 我們可以得出，我們提出的獎勵函數有比原始獎勵函數表現更好。並且使用本論文提出的 Baseline 策略來獲取獎勵的方式能夠得到最好的 Makespan。根據此實驗結果，在後續的實驗中會使用 Baseline 策略做為獎勵函數。

4.6 實驗三

為了驗證我們提出的方法是否比現有最強的 PDR 表現更好，我們設計實驗三來回答 RQ3。實驗三將分為 4 小節進行實驗，前 3 小節將從 L2D 的架構進行修改，確認修改後的架構是否優於原先架構。最後 1 小節將會使用在 JSSP 上表現強大的 4 個 PDR[1]與我們提出的方法在 DFJSP 上進行比較。

4.6.1 去除批標準化是否有效

GIN 的神經網路全連接層的 batch size 為圖形節點數量，由於我們在訓練環境中，圖形大小會改變，因此 batch size 是不固定的。原有的全連接層在進行激活函數之前有先進行批標準化（Batch Normalization，簡稱 BN），考慮到不固定的 batch size 可能影響神經網路的效能，因此本小節將實驗移除 BN，在兩種激活函數：修正線性單元（Rectified Linear Unit，簡稱 ReLU）和雙曲正切（Hyperbolic Tangent，簡稱 Tanh），與兩種資料前處理時所做的標準化：除以常數 1000 和將資料縮放至平均值 0 標準差 1 的範圍，進行組合，驗證哪一種方式能獲得最好的學習曲線。實驗結果如圖 4-6 所示。

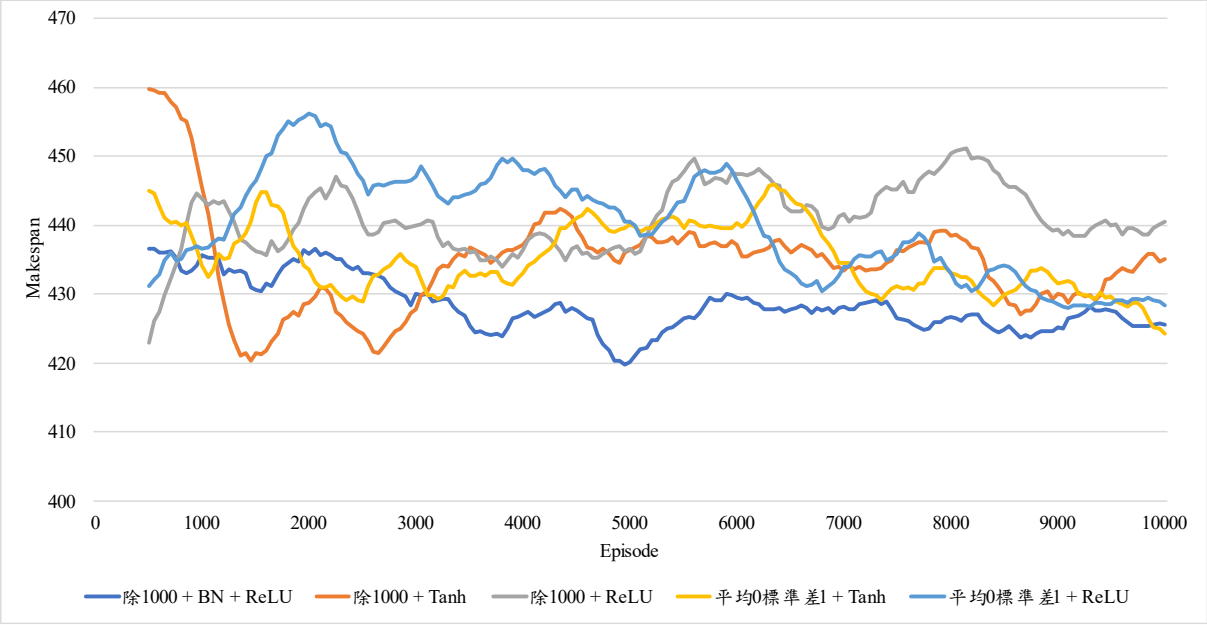

圖 4-6 不同激活函數在移除批標準化的 9×3 DFJSP 下之學習曲線

圖 4-6 使用原始獎勵函數進行學習，並將學習曲線以 500 個 episode 做移動平均。從實驗結果可以得知，使用除以常數 1000 並搭配 BN 和 ReLU 能得到最好的學習曲線，這種方法為原本 L2D 架構中所使用，因此我們不需要在 GIN 中移除 BN。

4.6.2 修改神經元數量是否有效

考慮到模型在收斂後對於測試集仍然與最佳解有一段差距，可能是模型過於簡單，導致擬合不足（Underfitting）、預測能力下降、訊息丟失、無法適應多樣化的問題。為了克服這些問題，可以通過增加模型的層數、增加神經元的數量、引入更多的特徵或使用更強大的學習演算法等方式來實現。此小節將修改神經元數量來驗證是否對於模型有幫助。

原始方法在 GIN 的隱藏層神經元數量為 64，PPO 的隱藏層神經元數量為 32，假設我們將此神經元數量定義為 1 倍，那麼我們將以 2 倍和 4 倍進行實驗。表 4-9 顯示了不同神經元數量在訓練中的 loss、value loss 和 episode reward。使用原始獎勵函數進行學習，訓練總共為 10000 episode，平均值和標準差皆取自範圍 5000 到 10000 episode。episode reward 表示在一個 episode 累積總合的 reward。從表中可以得知，神經元數量越高，loss 和 value loss 越低，但是 episode reward 在神經元數量為 2 倍時擁有最高的平均值。

表 4-9 不同神經元數量在 9×3 DFJSP 下的 Loss 和 Episode Reward

		神經元數量		
		1 倍	2 倍	4 倍
loss	平均值 μ	0.5754	0.4785	**0.4552**
	標準差 σ	0.7190	**0.6428**	0.6639
value loss	平均值 μ	0.6665	0.6063	**0.5357**
	標準差 σ	0.2453	0.2393	**0.2066**
episode reward	平均值 μ	-213.8	**-210.6**	-212.9
	標準差 σ	35.8	35.7	**34.3**

圖 4-7 為表 4-9 的模型之學習曲線，學習曲線以 500 個 episode 做移動平均。從實驗結果可以得知，在神經元數量為 2 倍時，擁有最好的學習曲線，學習曲線有持續往下降，逐漸獲取較好的 Makespan。因此本論文將採用 2 倍神經元數量做為實驗設置。

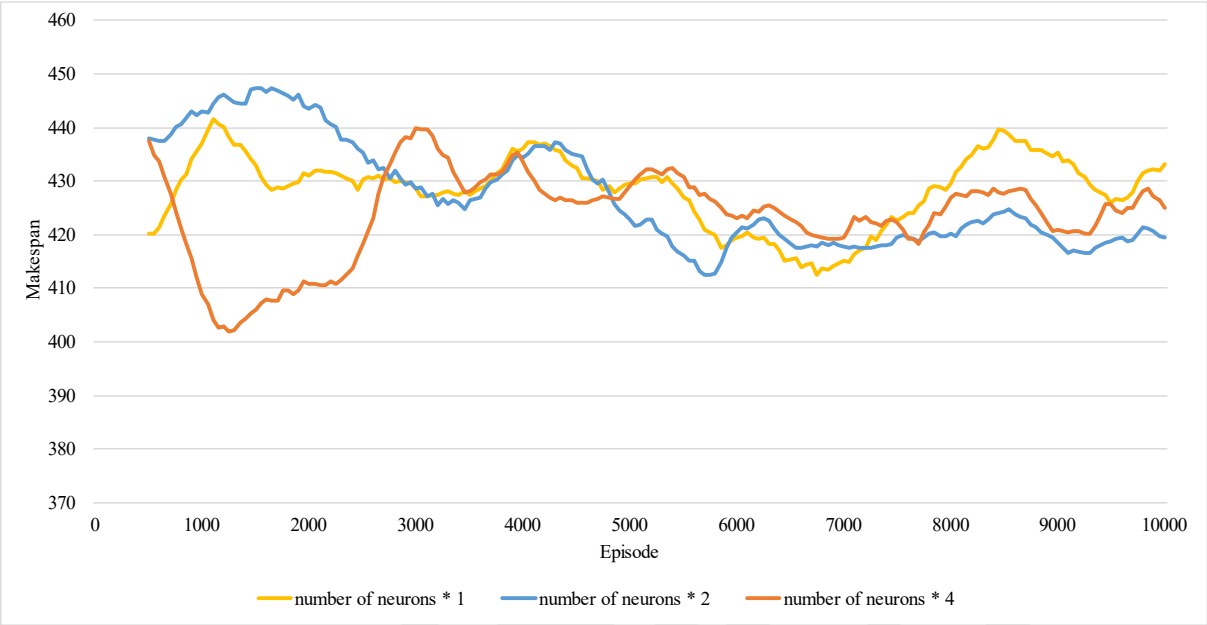

圖 4-7 不同神經元數量在 9×3 DFJSP 下的學習曲線

4.6.3 圖形池化是否有效

根據 3.3 節所述，我們將實驗對第一種圖形表示方法套用我們提出的圖形池化，是否能夠帶來更好的學習曲線。從圖 4-8 的實驗結果所示，雖然我們提出的方法能夠在訓練前期快速學習到策略，不過隨著訓練的推移開始出現過度擬合的情況，直到收斂於與原本圖形池化方法在同一條 Makespan 水平線上。我們認為該結果與圖 4-2 類似，進行圖形池化後，使嵌入特徵缺少了部份圖形的資訊。以結果而言，使用提出的方法並不會導致效能變差，反而有助於在早期快速學習策略，因此我們認為可以採用此方法。

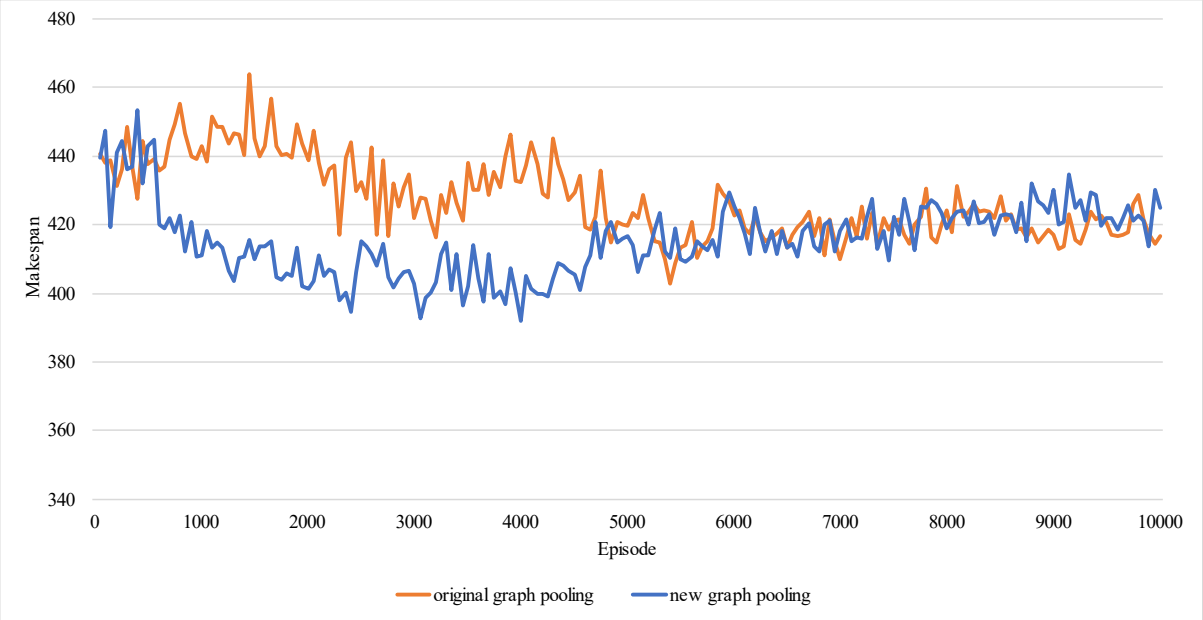

圖 4-8 使用提出的圖形池化方法在 9×3 DFJSP 下的學習曲線

4.6.4 在 DFJSP 中與不同 PDR 進行比較

在提出 L2D 方法的論文中[1]選擇了四種傳統的 PDR 進行比較，包含最短處理時間（SPT）、最多剩餘工作（MWKR）、流程到期日與最多剩餘工作的最小比率（FDD/MWKR）以及最多剩餘操作（MOPNR），其中 SPT 是工業界廣泛使用的 PDR 之一，其餘都是在 Taillard 測試集上表現最好的 PDR。因此本實驗將使用這 4 種 PDR 與我們提出的方法（以 DRL 表示）進行比較。為了加快學習收斂速度，本節將使用第二種圖形表示。

在 DFJSP 當中，同個任務可以有多種機器選擇，而這些 PDR 都只有針對任務的優先度規則，而沒有針對機器的優先度規則。雖然選擇機器的部分可以使用隨機選擇來處理，但是這不符合 PDR 的原則，PDR 應盡量減少隨機選擇的情況。因此我們將為這些 PDR 添加上第二個規則：選擇最早開始時間的機器，這會讓 DFJSP 產生較為緊湊的排程，從而獲得更好的 Makespan。

我們以兩種尺寸 6x3 和 15x5 的 FJSP 資料集訓練了兩種模型，使用該模型去預測相同尺寸的測試集，並與 4 種 PDR 和 OR-Tools 進行效能評估，結果如表 4-10 所示。

表 4-10 不同方法在 DFJSP 下的效能

Size	Avg. M_i		PDR				DRL	OR-Tools
			SPT	MWKR	FDD/MWKR	MOPNR		
6×3	2	Makespan	507.8	375.7	362.3	368.0	**342.5**	313
		Gap	62.2%	20.0%	15.8%	17.6%	**9.4%**	0.0%
		Time (s)	0.037	0.026	0.025	**0.024**	0.086	0.081
		Reschedule	9.4	6.5	5.9	5.9	5.3	0.0
		Opt. Rate						100%
15×5	2.58	Makespan	1373.1	736.6	704.7	717.7	**679.7**	560.6
		Gap	144.9%	31.4%	25.7%	28.0%	**21.2%**	0.0%
		Time (s)	1.501	0.768	0.730	**0.690**	1.655	13.445
		Reschedule	47.7	24.5	22.7	22.0	21.1	0.0
		Opt. Rate						100%

Avg. M_i 是所有機器類型初始機器數量的平均數，計算公式見第(4.1)式。舉例來說，Size 為 6×3 的 DFJSP，工作數量為 6，每個工作都有 3 個任務，而機器類型有 3 種，每種機器類型的機器初始數量為 $M_i \pm 20\%$，發生動態機台數量增減事件時，會使機器數量在 1 到個別機器類型的初始數量之間做變化。因此可以透過 Avg. M_i 了解到這個問題中的總體機器數量。Gap 為該方法與 OR-Tools 求出的最佳解之差距。Time (s) 的單位為秒，表示該方法平均求解一個 DFJSP 實例所花費的時間，由於 OR-Tools 會在某些實例下運算很久，因此這裡會限制 OR-Tools 單一實例求解時間上限為 600 秒。

Reschedule 表示該方法平均求解一個 DFJSP 實例所需要重新排程的次數，亦即動態事件的發生次數。因為真實情況中，我們無法得知 DFJSP 下一個動態事件會發生在哪一個時間點，所以需要根據目前的工作和機器進行完整一次的排程，之後才獲取下一個時間點的動態事件。動態事件會增加新機器或刪減現有機器，因此需要重新排程在此時間點以後的所有工作。如果剛好在這個時間點上有機器正在處理任務，若它不是被刪減的機器，則此任務則繼續處理；否則該任務需要中斷並且重新處理，交由重新排程來分配機器。動態事件發生次數等同於重新排程的次數，而 OR-Tools 的求解方式是先用 DRL 求解過一次，得到每台機器確定的加入時間和退役時間，再透過 DRL 所得到的確定時間，將 DFJSP 轉成靜態問題再由 OR-Tools 求解。所以 OR-Tools 沒有動態事件，故其 Reschedule 的次數 0。真實情況中只有發生動態事件時才需要重新排程一次，若想獲得該方法平均每一個實例中排程一次所花費的時間，其計算公式為 $\frac{\text{Time (s)}}{\text{Reschedule} + 1}$。

Opt. Rate 為 OR-Tools 在測試集中求解實例能在限制時間內取得最佳解的比例。例如測試集有 30 筆 DFJSP 實例，其中有 27 筆實例在 600 秒內取得最佳解，Opt. Rate 即為 90%。

從表 4-10 的實驗結果可以得知我們提出的方法 DRL 能比現有最強的 PDR 表現還要好，而花費最少時間的方法是 MOPNR。雖然我們的方法比 PDR 還要慢，但考慮在不同尺寸下，DRL 花費時間的時間複雜度與 PDR 的差距不大，因此是屬於在可容忍範圍之內。而 OR-Tools 在較大尺寸的實例下，花費時間有明顯大量的增加。

不同方法在尺寸較大的問題中 Gap 有明顯上升，我們認為造成此現象的原因除了問題變得複雜之外，由於無法預知機器數量何時會發生增減，並且機器增減的數量也是不可預知的，導致每一次的重新排程都會讓機器產生一些空閒時間，進而拉大了差距。

根據上述的實驗結果，對於 RQ3 我們可以得出，我們提出的方法在不同尺寸下的 DFJSP 都能比現有最強的 PDR 表現還要好。在 4 種 PDR 中，以 FDD/MWKR 表現最好。在 6×3 尺寸下，我們方法的 Gap 與 FDD/MWKR 相比能減少 6.3%；在 15×5 尺寸下，能減少 4.5%。在花費時間上則是 MOPNR 花費最少。

4.7 實驗四：以訓練好的模型來處理不同尺寸的 DFJSP

為了驗證訓練後的模型是否可以解決不同大小的 DFJSP 實例，我們設計實驗四來回答 RQ4。我們會從 4.6.4 小節中，以 15×5 實例大小的 FJSP 所訓練出來的模型，用於處理不同工作尺寸和機器尺寸（指每工作的任務數和機器類型數量）下的 DFJSP。評估其 Makespan 是否能夠保持原有的效能水平。使用 15×5 的模型在不同尺寸 DFJSP 實例的效能如表 4-11 和表 4-12 所示。

表 4-11 DFJSP 中使用 15x5 的模型在不同尺寸下的效能

Size	Avg. M_i		PDR				DRL	OR-Tools
			SPT	MWKR	FDD/MWKR	MOPNR		
6×3	2	Makespan	507.8	375.7	362.3	368.0	**342.8**	313
		Gap	62.2%	20.0%	15.8%	17.6%	**9.5%**	0.0%
		Time (s)	0.037	0.026	0.025	**0.024**	0.086	0.081
		Reschedule	9.4	6.5	5.9	5.9	5.3	0.0
		Opt. Rate						100%
15×3	3.32	Makespan	748.0	489.6	465.3	483.3	**460.3**	400.6
		Gap	86.7%	22.2%	16.2%	20.6%	**14.9%**	0.0%
		Time (s)	0.209	0.139	0.129	**0.128**	0.298	1.982
		Reschedule	11.3	7.2	6.4	6.3	5.8	0.0
		Opt. Rate						100%
30×3	4.32	Makespan	1224.8	786.3	784.9	797.4	**763.2**	684.7
		Gap	78.9%	14.8%	14.6%	16.5%	**11.5%**	0.0%
		Time (s)	0.948	0.661	0.657	**0.626**	0.872	600.000
		Reschedule	14.5	9.2	9.0	8.6	8.1	0.0
		Opt. Rate						0%
60×3	5.32	Makespan	1850.0	1160.6	1123.7	1143.5	**1115.9**	1149.1
		Gap	61.0%	1.0%	-2.2%	-0.5%	**-2.9%**	0.0%
		Time (s)	4.706	3.042	3.079	**2.985**	4.680	600.000
		Reschedule	18.7	11.4	11.1	10.7	10.3	0.0
		Opt. Rate						0%

續下頁

表 4-12 DFJSP 中使用 15x5 的模型在不同尺寸下的效能(續)

Size	Avg. M_i		PDR				DRL	OR-Tools
			SPT	MWKR	FDD/MWKR	MOPNR		
6×5	1.26	Makespan	891.8	597.3	563.1	560.6	**518.5**	448.6
		Gap	98.8%	33.1%	25.5%	25.0%	**15.6%**	0.0%
		Time (s)	0.167	0.108	0.100	**0.099**	0.329	0.145
		Reschedule	23.5	15.7	14.9	14.4	12.9	0.0
		Opt. Rate						100%
15×5	2.58	Makespan	1373.1	736.6	704.7	717.7	**679.7**	560.6
		Gap	144.9%	31.4%	25.7%	28.0%	**21.2%**	0.0%
		Time (s)	1.501	0.768	0.730	**0.690**	1.655	13.445
		Reschedule	47.7	24.5	22.7	22.0	21.1	0.0
		Opt. Rate						100%
30×5	3.58	Makespan	1915.3	1047.9	988.9	1004.5	**955.9**	844.2
		Gap	126.9%	24.1%	17.1%	19.0%	**13.2%**	0.0%
		Time (s)	5.612	3.056	2.918	**2.774**	5.104	600.000
		Reschedule	49.1	26.3	24.6	23.7	22.6	0.0
		Opt. Rate						0%
60×5	4.58	Makespan	3024.2	1551.7	1503.9	1501.7	**1451.9**	2735.1
		Gap	10.6%	-43.3%	-45.0%	-45.1%	**-46.9%**	0.0%
		Time (s)	26.729	13.683	13.285	**12.966**	19.495	600.000
		Reschedule	62.2	31.1	29.6	28.8	27.5	0.0
		Opt. Rate						0%
6×6	1	Makespan	1116.9	661.1	628.0	646.3	**570.3**	489.5
		Gap	128.2%	35.1%	28.3%	32.0%	**16.5%**	0.0%
		Time (s)	0.014	0.014	**0.013**	**0.013**	0.062	0.034
		Reschedule	0.0	0.0	0.0	0.0	0.0	0.0
		Opt. Rate						100%
15×10	1.58	Makespan	3391.2	1333.3	1279.6	1240.2	**1209.3**	918.4
		Gap	269.3%	45.2%	39.3%	35.0%	**31.7%**	0.0%
		Time (s)	21.757	7.613	7.173	**6.740**	16.921	124.383
		Reschedule	229.7	87.5	83.0	78.6	76.6	0.0
		Opt. Rate						100%
30×10	2.58	Makespan	4249.8	1613.9	1532.9	1493.6	**1466.1**	1358.0
		Gap	212.9%	18.8%	12.9%	10.0%	**8.0%**	0.0%
		Time (s)	75.657	25.667	24.342	**23.099**	34.357	600.000
		Reschedule	318.9	117.8	111.1	104.7	103.7	0.0
		Opt. Rate						0%
60×10	3.58	Makespan	6327.3	2301.9	2229.7	2107.1	**2086.2**	4490.6
		Gap	40.9%	-48.7%	-50.3%	-53.1%	**-53.5%**	0.0%
		Time (s)	449.958	162.277	150.711	**140.851**	203.364	600.000
		Reschedule	346.3	122.5	118.6	109.9	109.6	0.0
		Opt. Rate						0%

從實驗結果得知，不同大小的工作尺寸和機器尺寸下的 DFJSP，我們提出的方法都能比 PDR 表現還要好，花費最少時間的方法依然是 MOPNR。

我們先從不同的工作尺寸來看，6×5、15×5、30×5 和 60×5，隨著尺寸增加 OR-Tools 需要花費的時間呈指數式增長，從 30×5 開始 Opt. Rate 已經降為 0，在 60×5 的時候排程的 Makespan 很差，意味著 OR-Tools 無法處理大尺寸問題，因為機器數量變多，讓 OR-Tools 要探索的路徑空間變得太過龐大。而 DRL 和 PDR 之間的差距並無太大變化，說明 DRL 所學到的策略可以應用到不同工作尺寸上，能具有良好的泛化能力，不會使表現變得太差。

從不同的機器尺寸來看，15×3、15×5 和 15×10，以 PDR 來說可以明顯看出 FDD/MWKR 在 15×3 表現好，而到 15×10 則變成 MOPNR 表現好，說明 FDD/MWKR 在同種類機器數量多的時候有效，而 MOPNR 在同種類機器數量少的時候有效，驗證了 PDR 只能在特定情況下有效。隨著機器尺寸變化，DRL 與最好 PDR 的 Gap 差距從 1.3% 到 3.3%，雖然當同種類機器數量越多時，DRL 與最好 PDR 的差距會逐漸縮小，但仍然能夠優於 PDR，因此 DRL 模型的泛化能力有一定程度上可以適應這種同種類機器數量的變化。

接下來我們要驗證不同方法在特定數量範圍內的動態機器數量 FJSP 下的效能。針對該實驗，測試集與訓練集尺寸相同。我們會限制每種類的機器上限和下限，該值由表 4-13 中的 max 和 min 給出。其餘的規則都皆依照 4.2 節的規則。

實驗結果如表 4-13 所示，機器數量上下限皆為 1 的問題是屬於 JSSP，不會出現動態事件，因此僅列在表中做為參考。隨著機器數量上限的提高，FDD/MWKR 表現得更有優勢，在機器數量上限為 4 的情況下，DRL 會輸給 FDD/MWKR，意味著在固定工作尺寸下，DRL 較不擅長處理同種類機器數量多的問題。

表 4-13 DFJSP 中使用 15x5 的模型在不同機器數量下的效能

Size: 15×5 machines min	max		PDR SPT	PDR MWKR	PDR FDD/MWKR	PDR MOPNR	DRL	OR-Tools
1	1	Makespan	2117.5	1175.9	1138.9	1125.5	**959.3**	901.3
		Gap	134.9%	30.5%	26.4%	24.9%	**6.4%**	0.0%
		Time (s)	0.048	0.046	**0.045**	0.047	0.152	0.103
		Reschedule	0.0	0.0	0.0	0.0	0.0	0.0
		Opt. Rate						100%
1	2	Makespan	1531.5	820.5	796.4	764.3	**744.4**	603.9
		Gap	153.6%	35.9%	31.9%	26.6%	**23.3%**	0.0%
		Time (s)	1.627	0.810	0.810	**0.716**	1.804	15.652
		Reschedule	53.5	27.6	26.9	24.0	23.2	0.0
		Opt. Rate						100%
1	3	Makespan	1244.8	664.0	640.1	651.7	**617.2**	498.3
		Gap	149.8%	33.3%	28.5%	30.8%	**23.9%**	0.0%
		Time (s)	1.389	0.742	0.701	**0.686**	1.524	19.626
		Reschedule	42.9	23.9	20.5	19.8	18.7	0.0
		Opt. Rate						100%
1	4	Makespan	1026.1	568.2	**531.9**	549.2	548.9	436.5
		Gap	135.1%	30.2%	**21.9%**	25.8%	25.8%	0.0%
		Time (s)	1.201	0.672	0.624	**0.615**	1.398	25.535
		Reschedule	34.7	18.0	16.3	16.0	16.3	0.0
		Opt. Rate						100%
1	5	Makespan	892.4	511.9	**498.4**	514.3	511.5	420.4
		Gap	112.3%	21.8%	**18.6%**	22.3%	21.7%	0.0%
		Time (s)	1.132	0.621	0.602	**0.597**	1.274	21.332
		Reschedule	29.3	15.5	14.8	14.5	14.3	0.0
		Opt. Rate						100%

表 4-14 顯示了不同方法在 15x5 的靜態 FJSP 下的效能。num of machines 表示每一種類的機器各有多少台。從實驗結果可以得知，在靜態問題下，我們提出的方法在 FJSP 機器數量為 3 以下能夠超越 PDR。MWKR 和 FDD/MWKR 在機器數量較多的時候表現較好。

表 4-14 不同方法在 FJSP 下的效能

Size: 15×5 num of machines		PDR				DRL	OR-Tools
		SPT	MWKR	FDD/MWKR	MOPNR		
1	Makespan	2117.5	1175.9	1138.9	1125.5	**959.3**	901.3
	Gap	134.9%	30.5%	26.4%	24.9%	**6.4%**	0.0%
	Time (s)	0.048	0.046	**0.045**	0.047	0.152	0.103
	Opt. Rate						100%
2	Makespan	1102.2	583.9	569.3	595.0	**533.9**	462.3
	Gap	138.4%	26.3%	23.1%	28.7%	**15.5%**	0.0%
	Time (s)	0.056	0.055	**0.054**	0.056	0.197	139.523
	Opt. Rate						83.3%
3	Makespan	808.6	421.5	419.4	445.2	**417.9**	374.6
	Gap	115.9%	12.5%	12.0%	18.8%	**11.6%**	0.0%
	Time (s)	0.062	0.063	**0.061**	**0.061**	0.203	46.523
	Opt. Rate						96.7%
4	Makespan	659.0	**369.4**	378.3	391.6	385.9	368.3
	Gap	78.9%	**0.3%**	2.7%	6.3%	4.8%	0.0%
	Time (s)	0.072	**0.069**	0.073	0.070	0.207	13.764
	Opt. Rate						100%
5	Makespan	583.6	**368.1**	368.4	372.9	370.8	368.1
	Gap	58.5%	**0.0%**	0.1%	1.3%	0.7%	0.0%
	Time (s)	**0.072**	0.077	0.077	0.085	0.231	1.258
	Opt. Rate						100%

根據上述的實驗結果，對於 RQ4 我們可以得出，訓練後的模型可以解決不同大小的 DFJSP 實例。我們能夠用一個小尺寸的模型來處理各種大尺寸的實例。在不同工作尺寸和機器尺寸下，都能保持良好的泛化能力。

4.8 討論

關於我們提出的方法相比原論文[1]的實驗結果，中小型 JSSP 實例（6×6 至 30×20）的 L2D 與 OR-Tools 的 Gap 差距為 17.7% 至 29.2%；我們的方法在 15×5 的 JSSP 中為 6.4%，DFJSP 為 21.2%，因此效能並沒有變得比較差。原論文最好的 PDR 在 JSSP 中的 Gap 差距落在 24.0% 至 48.6%；而在我們的 DFJSP 中落在 25.7%，並根據表 4-13 和表 4-14 的結果，可以認為 PDR 在多台機器上的效能比起單台機器有顯著的成長。

我們提出的方法雖然具有良好的泛化能力，但是在機器數量多的問題上會輸給 PDR，因此我們認為 DRL 在相同類型機器的分配上能力較弱，添加與機器數量有關的特徵或許可以改善 DRL 這方面的弱點。

在現有的架構和狀態模型下，調整獎勵函數的效果很有限，因此我們認為代理想要進一步地學習，要從架構或狀態模型上開始解決。我們觀察圖形中每一個節點的特徵值與代理選擇動作的關係，在每一 episode 訓練環境的中後期之動作選擇，第一特徵「任務完成時間下限」的影響大於第三特徵「任務實際開始時間」，且特徵值越小動作被選擇的機率就越大。而在圖 4-3 中，不合適的特徵會降低狀態嵌入的效能，因此挑選一個合適的特徵有助於讓代理獲得更好的學習。

綜合上述而言，在不修改 L2D 架構的前提下，想繼續讓我們的方法變得更好，應該先從狀態模型開始下手，修改圖形表示方法以及添加更有用的特徵。

第五章 結論及未來研究方向

5.1 結論

本論文以 L2D 為基礎，提出了一種基於深度強化學習的動態調度框架，以最小化 DFJSP 中的 Makespan。我們的方法不僅可以處理靜態的 FJSP，也能夠處理動態機台數量的 DFJSP。該方法基於 GIN 狀態嵌入網路和動態的圖形表示，可以任意輸入不同尺寸的分離圖，以便將訓練好模型重新用於解決不同規模的實例。在本論文提出的動態規則下的 DFJSP，我們的方法能夠優於傳統的 PDR，並且在不同工作尺寸和機器尺寸下，都能保持良好的泛化能力。

5.2 未來研究方向

我們可以嘗試更多的特徵，例如添加與機器數量有關的特徵，改善 DRL 代理在相同類型機器分配上的能力。特徵不應只限定於累計數值，可以添加基於百分比的特徵，比如機器平均利用率、任務延遲比率，進一步提高該方法的效能。目前的方法中，DFJSP 發生動態事件只有反應在圖形結構上，代理不一定知道發生了什麼事件，從而無法正確的學習調度規則，最好是添加一個基於圖形特徵的事件指標，比如距離上次事件經過的時間長度或是發生的事件類型，能夠讓代理根據指標去學習合適調度規則。

我們還可以嘗試新的目標。目前我們只追求最小化 Makespan，未來可以將目標函數改成更為接近實際工廠生產中的目標，比如工作早於預定時間結束的時間長度、機器之間的工作負載平衡、時間內所得的產品利潤，以這些目標來做為目標函數。上述目標都能夠為生產製造業帶來更好的發展，幫助產業解決各式各樣的問題。

參考文獻

[1] Zhang, C., et al., *Learning to dispatch for job shop scheduling via deep reinforcement learning*, in *Proceedings of the 34th International Conference on Neural Information Processing Systems*. 2020, Curran Associates Inc.: Vancouver, BC, Canada. p. Article 137.

[2] Sels, V., N. Gheysen, and M. Vanhoucke, *A comparison of priority rules for the job shop scheduling problem under different flow time- and tardness-related objective functions.* International journal of production research, 2012. **50**(15, (1.8)): p. 4255-4270.

[3] Heger, J. and T. Voss, *Dynamically changing sequencing rules with reinforcement learning in a job shop system with stochastic influences*, in *Proceedings of the Winter Simulation Conference*. 2021, IEEE Press: Orlando, Florida. p. 1608–1618.

[4] Turgut, Y. and C.E. Bozdag, *Deep Q-network model for dynamic job shop scheduling pproblem based on discrete event simulation*, in *Proceedings of the Winter Simulation Conference*. 2021, IEEE Press: Orlando, Florida. p. 1551–1559.

[5] Zhang, Y., et al., *Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems.* Robot. Comput.-Integr. Manuf., 2022. **78**(C): p. 20.

[6] Chang, J., et al., *Deep Reinforcement Learning for Dynamic Flexible Job Shop Scheduling with Random Job Arrival.* Processes, 2022. **10**(4): p. 760.

[7] Lei, K., et al., *Large-Scale Dynamic Scheduling for Flexible Job-Shop With Random Arrivals of New Jobs by Hierarchical Reinforcement Learning.* IEEE Transactions on Industrial Informatics, 2023: p. 1-12.

[8] Lei, K., et al. *An End-to-end Hierarchical Reinforcement Learning Framework for Large-scale Dynamic Flexible Job-shop Scheduling Problem*. in *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022.

[9] jolibrain. *Wheatley*. 2023 6,30,2023; Available from: https://github.com/jolibrain/wheatley.

[10] Duan, L., et al., *Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach*, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, Association for Computing Machinery: Virtual Event, CA, USA. p. 3054–3063.