基於邊緣計算之智慧工廠下精粹深度學習法於零工式生產排程

陳芋勻[1] 吳和晏[1] 林春成[1,*]

[1]國立陽明交通大學工業工程與管理學系
*cclin321@nycu.edu.tw

摘要

在工業 4.0 的概念下，自動化在半導體製造中扮演關鍵角色，而有效的排程能減少浪費並提高系統利用率，因此排程是成本控制的關鍵。這項研究使用了多決策深度學習(Multi-Decision Reinforcement Learning，MDRL)作為基礎架構，融合了邊緣運算和雲端運算的智慧工廠架構。透過雲端的神經網路計算，為各個機台制定最佳的派工規則，並透過邊緣運算裝置進行監控，將機台運作資料傳回雲端，不只提供即時資訊並確保生產流程順暢，還是下一個學習週期的基礎。然而，MDRL 仍有改進的空間。因此，本研究旨在優化 MDRL，將其概念應用於靈活製造中的排程問題，進一步提高系統運行效率。優化的方法包括減少神經網路輸入層資料的複雜度，藉此減輕雲端運算裝置的負擔。此外，通過刪除實驗結果不佳的派工法則，整體效率得到顯著提升。同時，為提高結果準確度，本研究增加神經網路中狀態資料的分割密度，提高效率的同時保持資料精準度。

關鍵字：智慧工廠、零工生產問題、雲端計算、邊緣計算、多決策深度學習

1 緒論

晶圓代工廠的排程在製造過程中扮演關鍵的角色，有效的排程可以幫助管理者監控生產狀況，減少資源浪費，提高整體系統利用率。目前排程優化的研究著重於分散式智慧排程，根據半導體產品特性的多樣性，採用了多決策深度強化學習方法。然而，儘管這方法能貼合產品需求，仍有優化的空間。本研究將提出改善方案，進化排程系統。

從過去的相關研究中，已經有一些強化學習架構試圖改善 JSP 問題，目標是取得用時短且不存在重疊的排程結果。有 JSP 相關研究(Zhang et al., 1995)提出強化學習方法來學習特定領域的啟發式算法，並用於 Job shop scheduling 排程。該方

法利用了時間差異算法來訓練計算模型，並用於對每個狀態進行評估，進而產生評估函數。接下來在涉及少量工作的問題上進行訓練，最後在更大的問題上進行測試，證實強化學習可以提供構建高性能排程系統的新方法。

目前排程優化研究多著重分散式智慧排程，根據半導體產品特性的多樣性，採用了多決策深度強化學習方法(Lin et al., 2019)。然而，儘管這方法能貼合產品需求，仍有優化的空間。本研究將提出改善方案，進化排程系統。本研究之主要貢獻如下：

- 在保留機台收集資料排程的數據真實度，讓決策系統做出更精準的判斷。同時保留MDRL的精隨。
- 為了降低運算成本，本研究提出的架構縮減神經網路的輸入層參考資料，也針對效率明顯低落的派工法則進行刪減。除了能精簡整個系統的資料複雜度，還能減輕神經網路運算的負擔，進而達成降低成本的最終目的。

2 文獻回顧

2.1 智慧工廠、雲端計算與邊緣計算

排程系統的效率和效益是取得競爭優勢的關鍵因素。提高機台利用率能有效降低工廠資本成本，而排程正是優化機台利用率的核心。邊緣計算裝置的引入能夠減少訂單排程的反應延遲時間。根據(Yi et al., 2015)以及(Ha et al., 2014)的研究，邊緣計算裝置對提升效率有著顯著的影響。此外雲端計算也是熱門的研究對象，能夠快速指派運算結果及同時監控和量測各機台裝置的運行情況。

2.2 深度強化學習(Deep Reinforcement Learning)

深度學習(Deep Learning) (LeCun et al, 2015)與強化學習(Reinforcement Learning) (Sutton et al, 1998)兩者的結合便形成了深度強化學習(Deep Q-Learning)。與一般取最近一次的經驗學習不同，深度強化學習是利用先前的經驗進行隨機採樣學習。深度 Q 網路(Deep Q Network，DQN)的概念為設計出一神經網路，透過紀錄狀態並輸出每一行為的 Q 值，來減少記憶體中需儲存的資訊量。作為 RL 的一種延伸算法。

2.3 多決策強化學習方法(Multi-Decision Reinforcement Learning，MDRL)

於是本研究選用 MDRL，也就是同時具備邊緣計算裝置以及雲端計算裝置的

排程系統，以解決零工生產問題。但 MDRL 仍存瑕疵，也就是汲取過多非必要資訊，考量不具優化效益的派工法則，增加計算裝置的負擔，並且在數據統計切分方面，過度模糊導致無法精準呈現數據的真實性。總結來說，整體計算裝置的運算速度無法達到最佳表現。

3 研究方法

本研究之目標為利用本研究所提出的精簡深度強化學習(Refined Deep Q Learning, RDQN)來解決有雲端計算與邊緣運算架構之智慧工廠中的零工生產排程(如圖 1 所示)，為每部機台分配達最高效率的派工法則，並依此排定加工順序。雲端計算中心依客戶訂單的加工資訊為每部機台分配派工法則(如表 2 所示)，邊緣計算確定每部機台的排程結果後，將修正神經網路所需資訊回傳到雲端計算中心，雲端計算中心則基於回饋來調整神經網路，以用於下一次神經網路運算。

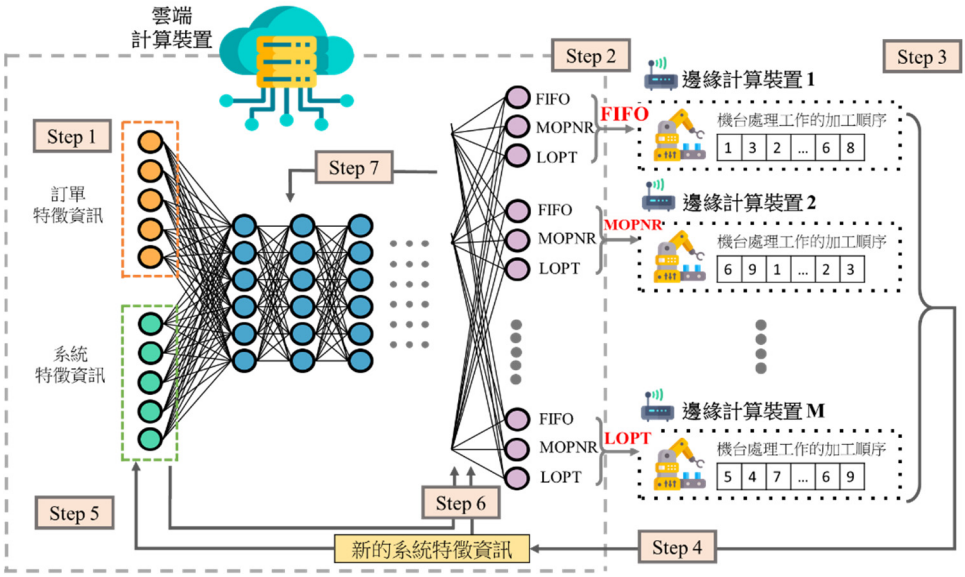

圖2、本研究所提出RDQN之流程圖

本研究所提 RDQN 與 MDRL 有下列幾點的差異：1) 派工法則的採用數量：在 MDRL 論文的基礎上，我們對派工法則的數量進行了刪減，針對其中四個工時間明顯較長的派工法則 SPT、LPT、SQN、LQN 刪除，只保留 FIFO、MOPNR、LOPT。減少神經網路進行運算的時間，精簡化派工法則的複雜度。2) 減少輸入層：同理，為了降低資料輸入所耗費的時間與程序，將輸入層的數量略作減少。3) 減少資料的失真：為了避免 C_{max} 與利用率失真，因此新增至 8 個區間，使資料不因區間數量而失去參考性。

表 2、本研究所使用之派工法則表

派工法則	優先條件
先進先出法 (First In First Out, FIFO)	越先早到達之工作越優先加工處理。
最長後續加工時間 (Most Operations Remaining, MOPNR)	後續作業加工時間越短越優先處理。
最多後續工作數 (Longest operation processing time, LOPT)	後續作業加工時間越長越優先處理。

本研究提出基於多決策深度強化學習的 RDQN 方法，能更有效率的解決智慧工廠的零工排程問題。以下將說明如何透過此方法訓練一個自動排程的模型，並將流程透過 Algorithm 1 表示。本研究所使用到的符號總結如表 3 所示，而所訓練的神經網路輸入層組成如表 4 所示。

Algorithm 1: MDRL

Input: Train set $\boldsymbol{T}_{\mathbf{train}}(1, 2, \ldots, N)$

Output: θ(Weight of neural network)

for $n = 1, 2, \ldots, N$ **do**
  Cloud center reads J jobs in customer order n and assigns a random or certain dispatching rule to each edge device
  Compute system features $\overline{F}, C_{\max}, \overline{Q}, \{\mu_m\}$, and $\{\overline{O}_m\}$
  Create a state based on (J, $C_{\max}$, μ_1, μ_2, …, μ_M) in the Q table
  Construct an NN with $8 + 2 \cdot M$ input neurons and $7 \cdot M$ output neurons
  for epoch number $t = 1, 2, \ldots, T$ **do**
    The cloud center conducts neural network forward propagation in the cloud center // **Algorithm 2**
    The cloud center determines dispatching rules of all machines, and transmit the information to each edge device// **Algorithm 3**
    Each edge device schedules all jobs in the controlled machine by the assigned dispatching rule // **Algorithm 4**
    The cloud center repairs the scheduling results that violates the research hypothesis.// **Algorithm 5**
    The cloud center creates a state based on (J, $C_{\max}$, μ_1, μ_2, …, μ_M) in the Q table, and uses $q'_m = [q_m + (\mu'_m - \mu_m)/\mu_m] + \gamma \cdot \mathrm{Max}_a\, Q[s', a]$ to update the Q table and the neuron values in the output layer
    The cloud center updates the system features: $\overline{F}, C_{\max}, \overline{Q}, \{\mu_m\}$, and $\{\overline{O}_m\}$
    Update the hidden layer by using the loss function $E[(Q'_m - Q_m)^2]$
  next for
next for

表 1、本研究所用之符號

符號	意義
N	訂單中的第 N 筆作業
T	訓練模型之迭代數(Epoch)
M	機台數量
J	工作數量
P_{jm}	工作 j 在機台 m 的加工時間
F_m	機台 m 的完工時間
C_j	工作 j 的完工時間
Q_{jm}	工作 j 在機台 m 的等候加工時間
μ_m	機台 m 的利用率
I_m	機台 m 的閒置時間
O_m	機台 m 的在製品數量
K_{jm}	工作 j 在機台 m 的完工時間

表 2、神經網路輸入層組成

客戶訂單特徵資訊	
機台數量	M
工單數量	J
總加工時間	$\sum_j \sum_m P_{jm}$
最大加工時間	$\mathrm{Max}_{j,m}\, P_{jm}$
最小加工時間	$\mathrm{Min}_{j,m}\, P_{jm}$
系統特徵資訊	
平均流程時間	$\bar{F} = (\sum_j C_j)/J$
總處理時間	$C\mathrm{Max}_j\, j_{max}$
工單平均等候處理時間	$\bar{Q} = (\sum_j \sum_m Q_{jm})/J$
各機台使用率	$\mu_m = (F_m - I_m)/F_m$
各機台在製品數量	$\bar{O}_m = (\sum_j K_{jm})/F_m$

以下為針對 RDQN 的 7 個步驟的詳細說明：

1) **Step 1 (在雲端中心的神經網路)**：如 Algorithm 2 所示，利用 J 個工作的訂單資訊，作為計算出訂單特徵資訊與系統特徵資訊的來源(第 3 行)。接著，將這些資訊送入輸入層(第 4 行)，最終在輸出層產生結果(本研究稱為 Q-value)。

2) **Step 2 (指派派工法則)**：如 Algorithm 3 所示，獲取到新的 Q-value 之後，雲端計算中心會根據輸出層提供的結果，將特定派工法則指派給所有的邊緣計算裝置。每一個機台均會被指派一種派工法則，由於此研究中有三個派工原則，因此每三個輸出層神經元就是一個機台。在本研究的 ε-greedy policy

中，ε設為 0.9，是隨機選一個派工法則的機率。而選擇擁有最大輸出值的派工法則的機率是 0.1，隨著訓練模型的迴圈次數變多，ε會隨著遞減。

Algorithm 2: Experimental Procedure
Input: Information of J jobs
Output: Q-values (neuron values in the output layer of neural network)

1: Compute $\sum_j \sum_m P_{mj}$, $max_{m,j} P_{mj}$, and $min_{m,j} P_{mj}$.
2: The order features M, J, $\sum_j \sum_m P_{mj}$, $max_{m,j} P_{mj}$, $min_{m,j} P_{mj}$ and the system features are fed into the neural network's input layer.
3: If this is the first epoch, the neural network's hidden layer will initialize its weights randomly.
4: Execute once of the neural network forward propagation.

Algorithm 3: Determining dispatching rules
Input: Q-value (neural network's output layer values)
Output: Dispatching rules for all machines

1: **for** m = 1 to M **do**
2: **if** $P \le \varepsilon$ **then**
3: Assign machine m with a random dispatching rule
4: **else**
5: Assign machine m with the dispatching rule corresponding to the largest output neuron value in the m-th group of output neurons
6: **end if**
7: **next for**

3) 利用派工法則排定機台的排程(Step 3)：如 Algorithm 4 所示，在第 m 台機台接收到雲端計算裝置所指派的派工法則後，每個邊緣計算裝置為機台進行排程。此外為了避免作業處理順序雷同的狀況，會依序利用 MONPR、SPT、Random 三種方法加以修正，最終生成的便是個別機台的加工處理順序。

4) 修正機台排程結果：如 Algorithm 5 所示，雖然前項的演算法可以有效的為各別機台機算出可行的排程結果，然而在雲端計算中心針對所有機台的排程進行宏觀性的監控時，容易發生作業於機台或機台於作業中有重疊的問題，因此需要對此進行修正。

5) 更新 Q 表與輸出層(Step 5)：我們將邊緣計算裝置連結的智慧工廠狀態定義了意下幾項：$(J, C_{\max}, \mu_1, \mu_2, \ldots, \mu_M)$， Q 表紀錄各個狀態轉換截至目前最佳的輸出行為結果，而後根據這些數據進行分配派工法則的依據。在狀態中 $C_{\max}$ 為工作完成的最大時間，而$\mu_1, \mu_2, \ldots, \mu_M$ 則代表各個機台在該狀態的利用率，兩者的值域皆頗為廣大。因此在進行模型訓練時為了縮短訓練所需時間，並盡量降低資料的失真程度，我們將兩者各切分為 8 類。

6) 更新系統特徵：接著會在根據新的系統特徵資料$(\overline{F}, C_{\max}, \overline{Q}, \{\mu_m\}, \{\overline{O}_m\})$以及更新過後的 Q 表，作為下一階段神經網路輸入層中的特徵值。

Algorithm 4: Determining the scheduling result of a machine

Input: The assigned dispatching rule of machine *m*

Output: Scheduling result of machine *m*

All edge devices schedule all jobs in each machine according to the assigned dispatching rule and it comes out all scheduling result
for $m = 1$ to M **do**
While the scheduling result based on the assigned rule is infeasible **then**
    if the scheduling result is infeasible **then**
        if the assigned rule is not MOPNR **then**
            Use the MOPNR rule to repair the repeated job numbers in the scheduling result
        end if
        if the scheduling result is infeasible **then**
            if the assigned rule is not SPT **then**
                Use the SPT rule to repair the repeated job numbers in the scheduling result
            end if
            if the scheduling result is infeasible **then**
                A random schedule result is adopted
            end if
        end if
    end if
next for

Algorithm 5 Repairing scheduling results

Input: The scheduling results of all machines

Output: A correct scheduling result for each machine

for $i = 1$ to J **do**
    for $j = 1$ to $M - 1$ **do**
        m_1 = index of the machine that processes task j of job i
        m_2 = index of the machine that processes task $(j + 1)$ of job i
        Let T_{im}^{s} denote the starting time and T_{im}^{e} denote the end time of job i at machine m according to the scheduling result of machine m
        $\delta_1 = T_{im_1}^{e} - T_{im_2}^{s}$
        if $\delta_1 > 0$ **then**
            $T_{im_2}^{s} = T_{im_2}^{s} + \delta_1$ and $T_{im_2}^{e} = T_{im_2}^{e} + \delta_1$
            Let j_2 denote the ordinal number in which job i is processed at machine m_2 according to the scheduling result of machine m_2, i.e., job i is the j_2-th job to be processed at machine m_2
            for each job k processed after job i at machine m_2 except for the last job **do**
                Let k' denote the index of the job processed next to job k at machine m_2
                $\delta_2 = T_{km_2}^{e} - T_{k'm_2}^{s}$
                if $\delta_2 > 0$ **then**
                $T_{k'm_2}^{s} = T_{k'm_2}^{s} + \delta_2$ and $T_{k'm_2}^{e} = T_{k'm_2}^{e} + \delta_2$
                end if
            next for
        end if
    next for
next for

7) 更新隱藏層：在最後則會再次回到第一步驟的驗算法中，將預測的 Q 值以及 Q 表中的 Q 值帶入損失函數：$E[(\acute{Q_m} - Q_m)^2]$，也就是 Q_m'減去 Q_m 的平方後之期望值作為隱藏層中權重更新的依據。

4 實驗結果

4.1 實驗環境

本研究中採用 job shop 指標資料庫中之資料集進行實驗，命名為「Lawrence (la)」、「Demirkol, Mehta, and Uzsoy (dmu)」、「Taillard (ta)」、「Applegate and Cook」。而模型中需要參數分別是神經網路中隱藏層裡面神經元的數量，ε-greedy 策略的參數，以及在更新 Q 表時使用的學習率。經過分析後得出最佳參數值分別是 NN = 100、$\varepsilon = 0.9$、ε-diff = 0.01、$\gamma = 0.7$。

4.2 參數分析

本階段會分析後續模型會需要的參數，並依據不同參數的收斂程度選擇最適當的最佳參數值。以神經網路中隱藏層裡面神經元的數量為例進行分析。在「la01」資料集中 $M = 5$ 及 $J = 10$ 的清況下，將起始值設定為 50。從圖 3 中可以看出，當神經元達到 100 時，整體的收斂效果相較其他數值佳，因此我們設定 100 為神經元數量最佳參數值。

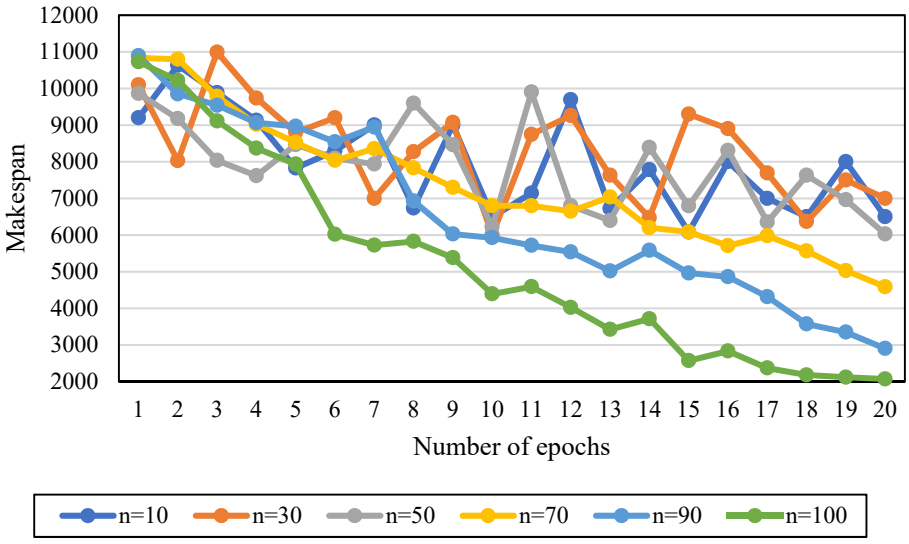

圖3、神經元數量分析

再以ε-diff 參數為例，從圖 4 中可知，此參數越小，模型的收斂速度越慢。而相較於其他測試值，起始值也就是 0.01 是最大的，我們便將此最佳參數值設為 0.01。

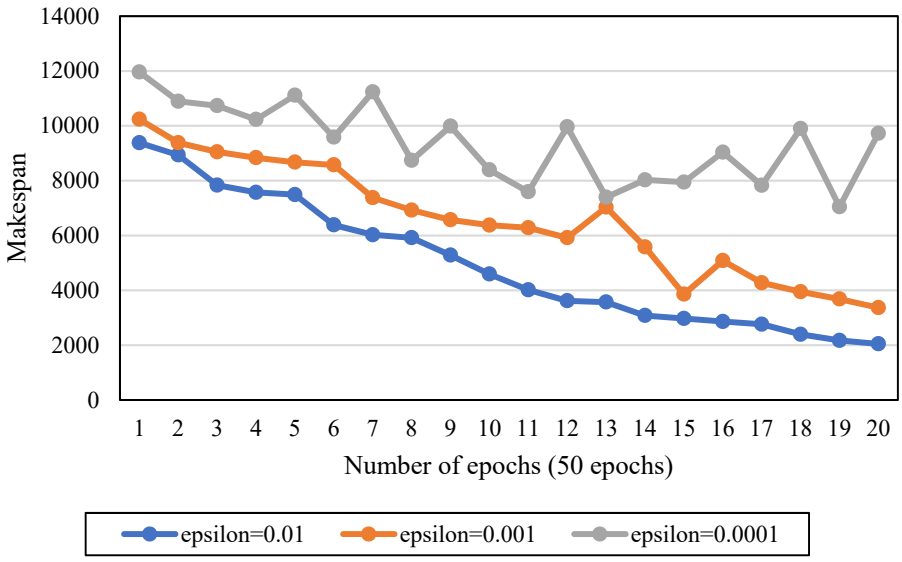

圖4、epsilon-diff參數收斂分析

4.3 實驗結果分析

本研究比較了 RDQN 與複合式派工法則及單一派工法則的訂單總完工時間。SDR 及 MDR 分別代表單一及複合式的派工法則，Random 則是間指派個機台處理加工順序。從表 6 中可知在機台數量為 15 的情況下，RDQN 的完工效率較 MDR 高出了 11.217%。

表 6、不同策略的計算時間比較

Instance	J	M	Random	RDQN	MDR		SDR	
					MDRL	FIFO	LOPT	MOPNR
ta06	15	15	11482	**1020**	1386	1496	1683	1736
ta07	15	15	12083	**1204**	1496	1693	1583	1739
ta08	15	15	10735	**1485**	1592	1830	1376	1736
ta09	15	15	11446	**1032**	1296	1482	1395	1303
ta10	15	15	11587	**1386**	1605	1704	1953	2074
ta16	20	15	20951	**1395**	1486	1583	1493	1624
ta17	20	15	19725	**1527**	1794	1936	2084	2104
ta18	20	15	19129	**1639**	1936	2385	2159	2275
ta19	20	15	18067	**1875**	2057	2204	2047	2493
ta20	20	15	19131	**1694**	1835	1957	1796	2041
ta36	30	15	34034	**2642**	2794	2947	2836	2815
ta37	30	15	32037	**2306**	2507	2749	2645	2603
ta38	30	15	34817	**2684**	2935	3085	2947	3195
ta39	30	15	33066	**2386**	2679	2794	3021	2905
ta40	30	15	36733	**2589**	2860	2974	3185	3075

5 結論

本研究所提出的架構適用於半導體的零工生產式智慧工廠。工廠會接收到來自客戶端的訂單資訊，將資訊傳入雲端計算裝置，並與邊緣計算裝置互動後，選擇出最適合該工作站的派工法則。本研究基於多決策深度強化學習(MDRL)的模型，進行效率的提升，分別是派工法則數量的刪減、增加狀態資料的密度、減少輸入層資料數量。依照實驗結果顯示，RDQN 模型在派工法則的決策效率相較 MDRL 有顯著的提升，尤其是當機台數量較多時，效率提升程度更佳。

關於未來展望，我們預期此研究模型可進行其他延伸。針對方法面向，未來研究可以調整 Q 表的計算方式、更改回饋值計算公式。而針對模型面向，未來研究可以延伸如在模型中加入霧計算，或將神經網路強化為 Double DQN 的架構。因此，此研究的後續應用範圍相當廣泛。

參考文獻

Applegate, D., and W. Cook (1991), "A computational study of the job-shop scheduling problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149-156.

Ha, K., et al. (2014), "You can teach elephants to dance: agile VM handoff for edge computing," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing (SEC 2017)*, article no. 12.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) "Deep learning," *Nature*, vol. 521, pp. 436-444.

Lin, C.-C., D.-J. Deng, Y.-L. Chih, and H.-T. Chiu (2019), "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4276-4284.

Sutton, R. S., and A. G. Barto (1998), "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no.5, pp. 1054-1054

Yi, S., C. Li, and Q. Li (2015), "A Survey of Fog Computing: Concepts, Applications and Issues," in *Proceedings of 2015 Workshop on Mobile Big Data (Mobidata 2015)*, pp. 37-42

Zhang, W., and T. G. Dietterich (1995), "A reinforcement learning approach to job-shop scheduling," in *Proceedings of 14th International Joint Conferences on Artificial Intelligence*, vol. 2, pp. 1114-1120.