

関西学院大学

2026年度 修士論文

遺伝的アルゴリズムと深層強化学習に基づく

ジョブスケジュール最適化問題

—PCB加工プロセスをモデルとして—

Job Scheduling Optimization Based on Genetic Algorithms and Deep Reinforcement Learning

—A Model of the PCB Manufacturing Process—

関西学院大学大学院

総合政策研究科 総合政策専攻

指導教員 山田 孝子 先生

学生番号 49024003

汪 永豪

2026年1月

---

## 要旨

近年, 市場グローバル化が進む中, 製品に対する市場ニーズは多様化しており, 顧客ごとに仕様が異なる多品種少量生産方式への対応が重要となっている。本研究では, プリント基板(PCB)製造工場を対象に, 同一加工機械を複数回使用する生産プロセスをモデル化した上で, ジョブショップスケジューリング問題に対する最適化手法について検討を行った。遺伝的アルゴリズム(GA)および深層強化学習に基づく Deep Q-Network(DQN)を用いてスケジューリングの最適化を行い, 両手法の性能比較を実施した。特に DQNにおいては, 複数の評価指標を組み合わせた複合型報酬関数と, *makespan* 中心型報酬関数をそれぞれ設計し, 両者のスケジューリング性能を GAとの比較を行った。その結果, 報酬関数の構成は最終的に得られるスケジュール品質に大きな影響を与えることを確認し, GA および DQN を用いた静的生産環境におけるスケジューリング最適化手法の性能を明らかにした。

キーワード：インダストリーエンジニアリング、オペレーション・リサーチ、ジョブスケジューリング最適化問題、遺伝的アルゴリズム、深層強化学習

---

# 目 次

要旨	1
目次	2
1. はじめに	4
2. 先行研究	6
3. 生産管理問題のモデル	8
3.1 モデル設定	8
3.2 問題設定	10
4. 実験用データ自動生成	12
4.1 多層 PCB の製造工程	14
4.1.1 6 製品種別の加工工程順序及び機械割当行列 $OM$ の構成	15
4.1.2 処理時間行列 $OT$ の構成	16
4.2 ジョブセットに含まれる注文データ生成	17
4.3 データ生成結果	19
5. 遺伝的アルゴリズムによる解の改善	21
5.1 遺伝的アルゴリズムの考え方	21
5.2 PPS 交差法の考え方	22
5.3 GA の実験結果	26
6. 深層強化学習による解の改善	31
6.1 深層強化学習の考え方	31
6.2 DQN を JSSP に適用する理由	36
6.3 報酬関数の構成	36

---

6.3.1	複合型報酬関数	37
6.3.2	<i>makespan</i> 中心型報酬関数	39
6.4	モデル訓練結果	40
6.4.1	複合型報酬関数での訓練	42
6.4.2	<i>makespan</i> 中心型報酬関数での訓練	45
6.5	訓練済みモデルでの実験結果	49
6.5.1	複合型報酬関数での実験結果	50
6.5.2	<i>makespan</i> 中心型報酬関数での実験結果	60
6.6	実験考察	72
7.	今後の課題	73
8.	謝辞	74

---

## 1. はじめに

世界経済のグローバル化が進展する中、製造業企業は国際的な競争環境にさらされ、従来以上に低コストかつ高品質な製品供給を求められている。特に、多品種少量生産への対応が不可欠となる現代の製造現場においては、生産設備を効率的に稼働させるための生産計画およびスケジューリングの高度化が重要な課題となっている。これらの課題に加え、安全管理や労務管理といった制約条件も考慮する必要があり、生産計画問題は年々複雑化している。

このような背景のもと、多くの製造現場では人手作業の代替として生産工程の自動化が進められており、プリント基板(PCB) 製造工場においても、複数の加工機械を繰り返し使用する複雑な生産プロセスが一般的となっている。このような生産環境は、典型的なジョブショップスケジューリング問題(JSSP)としてモデル化することができるが、JSSP は組合せ最適化問題であり、問題規模の増大に伴って計算量が急激に増加することが知られている。

JSSP に対する代表的な解法としては、遺伝的アルゴリズム(Genetic Algorithm:GA)をはじめとするメタヒューリスティクスが広く用いられてきた。一方で、近年では深層学習と強化学習を組み合わせた深層強化学習手法が注目されており、その一つである Deep Q-Network(DQN)は、逐次的な意思決定問題に対して有効な手法として知られている。

本研究では、PCB 製造工場を想定した生産プロセスをモデル化し、同一加工機械を複数回使用する JSSP を対象として、GA および DQN によるスケジューリング最適化を行う。特に DQN においては、複数の評価指標を組み合わせた複合型報酬関数と、*makespan* の最小化に着目した中心型報酬関数を設計し、報酬関数の構成が学習挙動およびスケジューリング性能に与える影響について比較・検討する。これにより、静的な生産環境における GA および DQN の特性を明らかにすることを目的とする。

本研究においてこれまでに取り組んだ内容を以下にまとめる。

1. 実験用ジョブセットとして、製品種別ごとの加工工程順序および加工時間の組み合わせデータを自動生成するシステムの構築
2. 生成されたジョブセットデータを用い、生産機械のスケジューリング問題に対して遺伝的アルゴリズムによる最適化実験

---

## ゴリズム (GA) により許容解を求める生産スケジュール案算出システムの構築

3. 得られた生産スケジュール結果を対象としたガントチャートによる可視化システムの構築
4. GA による最適スケジュール案の性能の評価
5. 深層強化学習に基づく Deep Q-Network(DQN) を用いたスケジューリング最適化手法の実装
6. DQN において、複合型報酬関数および *makespan* 中心型報酬関数を設計し、報酬構成の違いが学習挙動およびスケジューリング性能に与える影響の比較・検討

以上を踏まえ、本研究では、同一加工機械を複数回使用する生産プロセスを対象としたジョブショップスケジューリング問題 (Job Shop Scheduling Problem:JSSP) に対し、GA および DQN を用いたスケジューリング最適化手法の特性を明らかにすることを目的とする。特に、DQN における報酬関数設計の違いが得られるスケジュール品質に及ぼす影響について検討を行う。

---

## 2. 先行研究

グローバル経済の進展や企業のコスト削減要求に伴い、生産現場では柔軟で効率的な生産体制の構築が求められている。その中で、需要の多様化に対応するために、少量多品種生産が一般的となり、このような生産形態に適したスケジューリングの最適化が重要な課題となっている。特に、ジョブショップスケジューリング問題 (Job Shop Scheduling Problem: JSSP) は、生産システムにおける代表的な最適化課題として広く研究されている。JSSP は、複数のジョブを限られた機械に割り当て、全体の処理時間や納期遅延を最小化することを目的とする問題である。また、各ジョブには処理順序が定められており、この順序を厳密に守る必要がある点が特徴である。この問題は NP 困難であり、最適解を求ることは計算量的に非常に困難である。そのため、これまで多くの研究では、遺伝的アルゴリズム (Genetic Algorithm: GA) や焼きなまし法 (Simulated Annealing: SA) などのヒューリスティック手法を用いて、効率的に近似解を求めるアプローチが提案されてきた。しかし、実際の製造現場では、機械の故障や新規ジョブの追加、処理時間の変動など、動的因素が頻繁に発生する。そのような動的環境下では、あらかじめ定められたルールに基づくヒューリスティック手法では十分に対応できない場合がある。この問題を解決するため、近年では強化学習 (Reinforcement Learning: RL) を用いて、環境変化に応じて自律的にスケジューリング方策を学習する手法が注目されている。静的な環境においては GA や SA などの伝統的アルゴリズムが一定の成果を上げている一方で、動的な生産環境では強化学習に基づくアプローチのほうが、より高い適応性と柔軟性を示すことが報告されている。このように、JSSP におけるスケジューリング最適化の研究は、従来の探索型手法から学習型手法へと発展しつつある。

実際に、近年の研究ではこの流れを反映した多様なアプローチが報告されている。K.T. Chung ら [1] は現在工業 5.0 の時代に向けて、機械学習、特に強化学習を用いて、CO(combinatorial optimization) 問題を解決しようについて、現在の研究進展を述べた。M. Xu ら [2] は遺伝的プログラミングと強化学習を統合的に整理・比較し、JSSP スケジュール最適化問題において果たす役割と近年の研究動向を明らかにした。C. Ngwu ら [3] は動的な JSSP 問題は強化学習や進化的ヒューリスティック、機械学習を用いた手法でリアルタイムな意思決定は可能であることを示しました。X.R. Shao ら [4] は小規模かつ短時間の JSSP 問題に対して、多層畳み込みニューラルネットワーク (ML-CNN)

---

と反復局所探索 (ILS) を組み合わせ、ML-CNN で全体経路を学習し、ILS で最適な局所経路を探索する手法が提案した。L.B. Wang ら [5] は機械故障や再作業などの動的な環境での JSSP 問題について深層強化学習 (DRL) を用いた動的スケジューリング手法が提案され、PPO により最適な方策を学習することで、リアルタイムな生産スケジューリングが可能であることが示された。S. Lee ら [6] は射出成形金型製造プロセスにおいて、Deep Q-Network を活用することで従来より総加重遅延を低減できる金型生産スケジューリングを最適化した。C. Pickardt ら [7] は半導体製造プロセスにおいて、遺伝的プログラミング (GP) と確率的離散事象シミュレーションを組み合わせることで自動的に従来のルールより性能優れたなルールを生成した。村山ら [8] は JSSP 問題に基づき、加工機械と複数積載型 AGV の同時スケジューリングを対象とした遺伝的アルゴリズム (GA) 手法が提案した。

これらの研究から、JSSP およびその拡張問題に対して、静的環境ではヒューリスティック手法、動的環境では強化学習や深層学習手法が有効であることが確認されている。

---

### 3. 生産管理問題のモデル

本研究では受注生産を行う工場の製造工程, 受注型ジョブショップスケジュール問題 (Job Shop Scheduler prob-lem:JSSP) を最適化問題として解く. したがってモデルとしてクライアント側から提供される注文書が最適化の対象となり, このようなデータは不可欠である.

ここでは実際のプリント基板 (PCB) 製造工場をモデルの対象とする製造工程としてとりあげる. 本来は使用する注文書データが実生産に基づくものであることが望ましい. しかし, PCB 製造に関する実データは, 企業の営業機密や顧客情報に直結するデータのためこうしたデータは入手困難で, 共同研究であっても外部に具体的な数値の利用は難しい.

そのため本研究では, 先行研究 [22] に記載されている PCB 製造工程を参考にして, 各加工機械に対して加工役割を割り当てた人工的なデータをまず生成する.

本研究の最適化問題として解くのは, このデータを用いた PCB 製造プロセスの生産スケジューリング問題である. PCB 製造工程では, 注文種類によって必要とされる加工時間、加工工程数、あるいは加工順序が異なる. そこでこれらの差異を反映したデータを生成する.

以下で本論文の PCB 製造工程におけるモデル設定とモデルで使用する符号を定義する。

#### 3.1 モデル設定

1. 注文書は不特定の注文主から複数くる.
2. 各注文書には複数の製品種別が記載されている. ただし一つの注文書で, 製品種別が重複することはない.
3. 注文書には製品種別  $s$  ごとに注文するロット数が記載されている.
4. 加工工程の 1 ロットの製品単位は 10 個とする.
5. 加工工場は一定期間に様々な注文主からくる注文書  $O_i$  をまとめ, 同一出荷日をもつ注文書の束をジョブセットと呼ぶ.
6. 製品種別に応じて使用する加工機械、加工時間とその加工機械の使用順は予め決められている. ただし加工時間はロット数に依存する.

表 1: 本研究に用いる記号の定義

記号	説明
$O$	工場が受領した注文書の集合（またはジョブセットと呼ばれる）
$O_i$	第 $i$ 番に工場が受領した注文書（またはジョブセット $i$ と呼ばれる）
$J_s$	ジョブ（製品）でその製品種別が $s$ （またはジョブ $s$ と呼ばれる）
$O_i J_s$	第 $i$ 注文書に記載された製品種別 $s$ のジョブ
$lotJ_{i,s}$	第 $i$ 注文書に記載された製品種別 $s$ のジョブの注文ロット数（1 ロットの製品個数は 10）
$n_s$	対象ジョブセットに記載された製品種別 $s$ の総製造個数 $10 \sum_{i=1} lotJ_{i,s}$ (個数 = ロット数 × 10)
$J_s F_q$	製品種別 $s$ の第 $q$ 番目の加工工程
$m_{s,q}$	製品種別 $s$ の第 $q$ 工程に割り当てる機械番号
$M$	工場で稼働する加工機械の集合
$M_m$	第 $m$ 番目の加工機械
$N_{ops}$	ジョブセット内全種別製品の総加工工程の数
$t_{s,q}$	製品種別 $s$ が第 $q$ 工程に要する加工時間
$ft_{s,q}$	製品種別 $s$ の第 $q$ 工程の完了時刻
$ft_{m_{s,q}}$	機械 $m_{s,q}$ 上で第 $q$ 番目の工程の直前に処理された工程の完了時刻
$TJ_s$	製品種別 $s$ の加工終了時間
$makespan$	ジョブセットに含まれる全製品種別の最終工程完了時刻で最大値となる時刻
$OM$	各加工工程の機械割当を表す行列
$m_{s,q} = OM_{s,q}$	機械割当行列の要素定義
$OT$	各加工工程の加工時間を表す行列
$t_{s,q} = OT_{s,q}$	加工時間行列の要素定義
$C$	制約条件の集合

- 
7. ジョブセットごとに同一出荷日が設定されるが, 工場は出荷日が早いほどよいので, 受注した注文書の加工機械の利用順番をスケジューリング最適化し最短総加工生産時間を求める.
  8. 各機械はそれぞれ待機状態と稼働状態がある。

待機状態：機械は製品の加工作業していない状態にあり, いつでも新たな加工工程を開始できる.

稼働状態：機械は製品の加工作業している, 加工している作業が終わるまで, 新たな製品の加工工程は開始できない.
  9. 機械は注文書に記載された製品種別で指定されたロット数をまとめて加工する. 加工中に他の製品加工の割り込みはない.
  10. 機械のセットアップタイムは考慮しない.
  11. 製品の加工間の搬送時間はこのモデルでは考慮しない.
  12. 機械はあらかじめ与えた特定の加工工程のみ処理できる. 別加工工程への転用はできないものとする.
  13. モデルはイベント駆動型である.

イベント：加工工程の処理完了や機械の待機状態の発生など, スケジューリング状態が更新される契機となる事象を指す.
  14. ある製品の全加工工程が終了したら, 当該製品の加工スケジューリングは終了とする.
  15. ジョブセットに含まれる全製品の加工スケジューリングが終了したら, 当該ジョブセットのスケジューリングは終了する.

### 3.2 問題設定

本研究は JSSP を制約付き最適化問題として定式化し, 目標関数としては:

$$\text{opt. } JSSP = \min \left\{ \max_s T J_s \right\} \quad (1)$$

---

ここで,  $TJ_s$  は第  $s$  種別製品の最終加工工程の完了時刻を表す. したがって, 本最適化問題は, ジョブセットに記載された全製品種別の最終加工完了時刻の最大値を求める. そして様々なスケジューリング案の中で最終加工完了時刻の中で最小のスケジューリング案を本論文では *makespan* と呼ぶ, すなわち, 最適化とは, この *makespan* の最小化である. 最小となる *makespan* のスケジューリング案を作成することである.

ここで制約条件は :

$$\text{s.t. } C1: ft_{s,q} \geq ft_{s,q-1} + t_{s,q} \quad (2)$$

$$\text{s.t. } C2: ft_{s,q} \geq ft_{s',q'} + t_{s,q} \quad (3)$$

1. 制約条件  $C1$  は, 同一製品種別  $J_s$  における隣接する加工工程  $F_{q-1}$  と  $F_q$  の加工順序制約を表しており, 前工程が完了した後でなければ次工程を開始できないことを意味する.
2. 制約条件  $C2$  における  $ft_{s',q'}$  は加工工程  $J_s F_q$  と同じ機械  $m_{s,q}$  を使用するが, 製品種別が異なる別製品  $J_{s'}$  の加工工程  $J_{s'} F_{q'}$  の完了時刻を表す. すなわち,  $ft_{s',q'}$  は, 同一機械上で  $J_s F_q$  の直前に処理された工程の完了時刻に対応する.
3. 従って制約条件  $C2$  は, 各加工機械は同時刻に一つの加工工程しか処理できないという機械リソース制約を表しており, ある工程が機械  $M_m$  上で処理中の場合, 次の工程はその加工工程が終了するまで開始できない制約に対応する.
4. JSSPにおいては、加工工程と加工機械の対応関係を表す機械割当行列  $OM$  と, 各加工工程に要する加工時間を表す加工時間行列  $OT$  は予め既知なものとして, モデルでは与える. この  $OM$  と  $OT$  に関することは次章に述べる.

各製品種別の加工順序が与えられた場合, 各加工工程  $J_s F_q$  の完了時刻は次の式に基づいて算出することができる.

$$ft_{s,q} = \max (ft_{s,q-1}, ft_{m_{s,q}}) + t_{s,q} \quad (4)$$

---

ここで,  $ft_{s,q}$  は第  $s$  製品種別の第  $q$  加工工程の完了時刻を表す. 本式は, 各加工工程の完了時刻が, 同一製品における直前工程の完了時刻  $ft_{s,q-1}$  と, 同一加工機械  $m_{s,q}$  上で直前に処理された別製品工程の完了時刻  $ft_{m_{s,q}}$  のいずれか遅い方に, 加工時間  $t_{s,q}$  を加えた値として決定されることを示している.

ジョブショップスケジューリングの目的は各加工機械をできるだけ連続的に稼働させるとともに, 各製品の加工工程を待ち時間なく加工することにより, 総加工時間を最小化することである. したがって, JSSP は待機している加工機械に対して加工対象となるジョブを割り当てる逐次意思決定問題とみることができる. スケジューリングは加工機械上で各加工工程の処理完了イベントにごとに次に行う加工工程をジョブセットから選択することで逐次的に決まる. 同一製品内における加工工程の順序制約および、各加工工程が指定された加工機械で加工されなければならないという制約を考慮すると, 待機中の加工機械に製品の加工工程を割り当てるには, 現時点での処理可能な加工工程が存在するかどうかを判断する必要がある. 処理可能な加工工程がジョブセットに複数存在する場合には, その中から一つを選択して次の加工を開始する. 一方, 現時点で処理可能な加工工程が存在しない場合には, その加工機械は次の加工工程完了イベントが発生するまで待機状態となる. この意思決定過程は次の図 1 に示す.

## 4. 実験用データ自動生成

本来であれば, 行列  $OM$  および  $OT$  に用いる機械加工順序データや加工時間データは実工場から取得した実績データに基づくことが望ましい. しかしながら, 個々の製品の加工順序や加工時間、注文書などの生産情報は企業の機密および顧客情報として保護される. これらは共同研究等であっても外部公表は困難である. そこで本研究では、文献 [22] に記載された多層 PCB の製造工程を基づく各製品種別ごとの加工工程と使用機械をもとに, 注文書のばらつきやロット数を確率的にデータとして生成するし, ジョブセットを得る. 本節ではその実験データの自動生成について述べる. ジョブセットの生成手続きを以下に述べる.

まず最初に

1. 多層プリント基板の代表的な加工工程列を反映した機械割当行列  $OM$

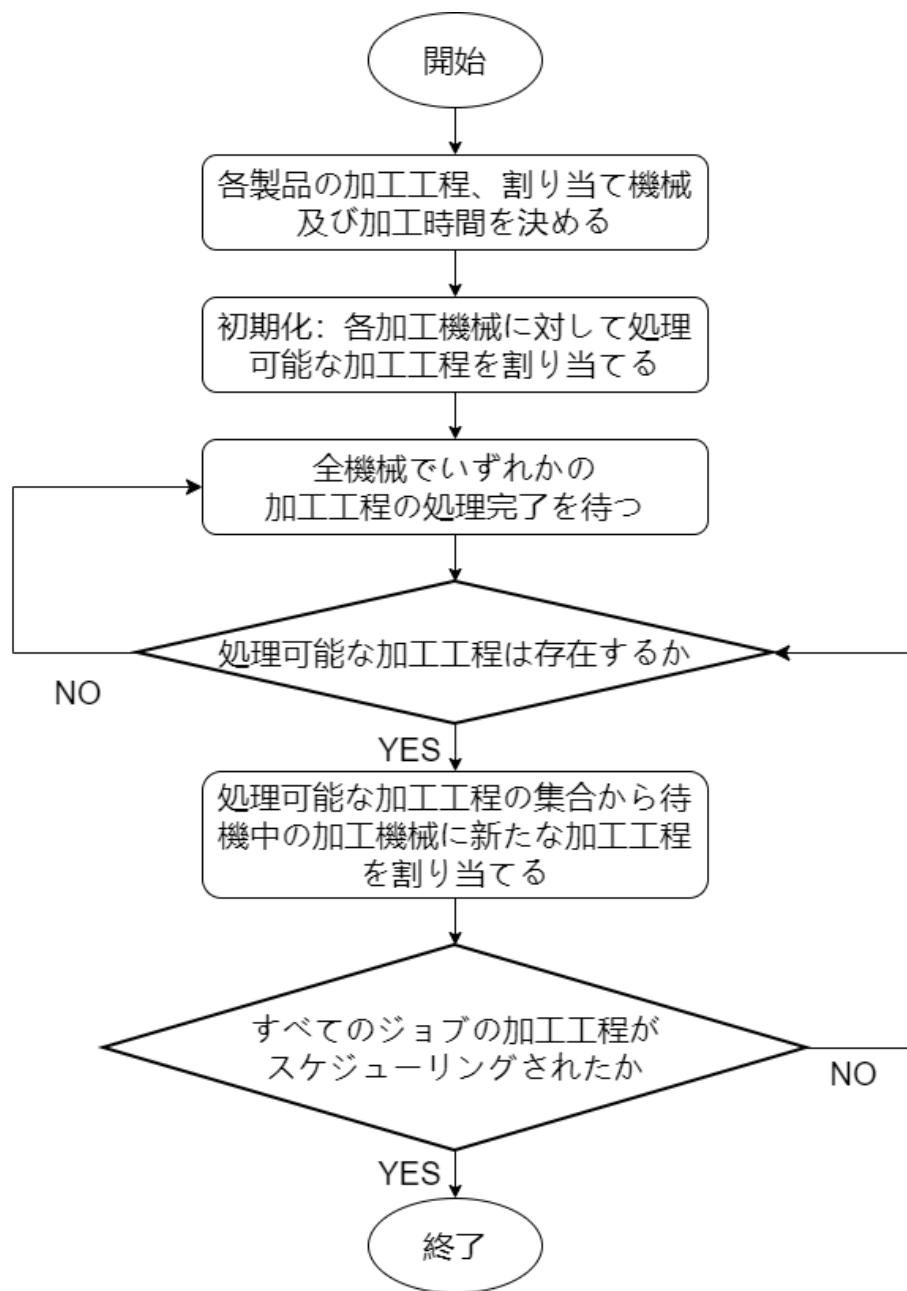


図 1: ジョブスケジューリング問題の逐次解を求める手続き

- 
2. 受注数・ロット構成のばらつきと機械特性を考慮して決定される処理時間行列  $OT$   
の二つの行列を用意する。次は行列  $OM$  と行列  $OT$  に入る要素について説明する

#### 4.1 多層 PCB の製造工程

多層 PCB の製造工程は以下の通りとする。

1. 材料準備
2. 材料表面の汚れや油分を除去し
3. パターン転写（フォトリソグラフィ方式かダイレクト印刷方式のいずれか一方とする）
4. エッチングとレジスト除去（複数回加工する場合ある）
5. 積層（複数回加工する場合ある）
6. ドリル加工
7. スルーホールめっき
8. ソルダーレジスト塗布
9. 表面処理
10. 電気検査
11. 包装

ここまで加工ができたら加工業は終了し出荷可能となる。この加工工程に対して、それぞれの工程に用いる加工機械の種類と役割を表2に示す。

本研究の生成データのため、製品種別として6種類の製品を生産する工場を仮定する。この6種類の製品の加工工程順序は以下のようにあらかじめ設定する。製品種別により各加工工程を一度しか要しない製品もあれば、複数回、同一加工工程を何度も他工程をはさみながら行き来する製品種別もある。

表 2: 加工機械および工程内容の定義

機械番号	工程内容
$M_1$	銅箔付きの絶縁板 1 を取る
$M_2$	銅箔付きの絶縁板 2 を取る
$M_3$	表面の汚れや油分を除去する
$M_4$	パターン転写 (フォトリソグラフィ方式)
$M_5$	パターン転写 (ダイレクト印刷方式)
$M_6$	エッティングおよびレジスト除去
$M_7$	積層
$M_8$	ドリル加工
$M_9$	スルーホールめっき
$M_{10}$	ソルダーレジスト塗布
$M_{11}$	表面処理
$M_{12}$	電気検査
$M_{13}$	包装

#### 4.1.1 6 製品種別の加工工程順序及び機械割当行列 $OM$ の構成

加工工場は 6 種類の製品種別に生産が可能で, 以下 1 行目から 6 行目までの各行を製品種別ごとの加工工程順序として工程番号を行列  $OM$  の形式で表す. 行は製品種別, 列は各製品の加工工程順番に対応する.

$$OM = \begin{bmatrix} 1 & 3 & 4 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 4 & 6 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 6 & 4 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 0 & 0 & 0 & 0 \\ 1 & 3 & 5 & 6 & 4 & 6 & 7 & 8 & 9 & 4 & 6 & 10 & 11 & 12 & 13 & 0 & 0 \\ 2 & 3 & 5 & 6 & 4 & 6 & 7 & 8 & 9 & 4 & 6 & 5 & 6 & 10 & 11 & 12 & 13 \end{bmatrix}$$

$OM_{s,q} = \{1, 2, \dots, 12, 13\}$  は製品  $s$  の第  $q$  番目加工工程の加工機械は  $OM_{s,q}$  の要素を意味する.

$OM_{s,q} = 0$  の場合, 製品  $s$  の第  $q$  番目加工工程自体がないことを意味する. こうして機械割当行列  $OM$  の構成が確定する。本研究はこの 6 種類の製品種別について, 最適スケジューリングを探すことで, 機械割当行列  $OM$  を変更することはしない, すなわち機械割当行列  $OM$  は固定された項目である。

### 4.1.2 処理時間行列 $OT$ の構成

処理時間行列  $OT$  は機械割当行列  $OM$  と違い、ジョブセットに含まれた注文書の個数や注文された製品のロット数により変更する。

エッチングとレジスト除去、積層、スルーホールめっき、この三つの加工操作は PCB をロット単位で一度に加工できる工程となる。したがってこの 3 つの工程の加工時間は定値とする。これ以外の加工工程ではその工程の加工時間は  $n_s \times$  単位時間の積となる。次は各機械の単位加工時間もしくは定値加工時間を設定する。

表 3: 加工機械別加工時間パラメータの設定

加工時間パラメータ	設定値
$MT_1$	0.5 (単位加工時間)
$MT_2$	0.5 (単位加工時間)
$MT_3$	2 (単位加工時間)
$MT_4$	8 (単位加工時間)
$MT_5$	7 (単位加工時間)
$MT_6$	250 (基準加工時間)
$MT_7$	600 (基準加工時間)
$MT_8$	3 (単位加工時間)
$MT_9$	750 (基準加工時間)
$MT_{10}$	5 (単位加工時間)
$MT_{11}$	4 (単位加工時間)
$MT_{12}$	3 (単位加工時間)
$MT_{13}$	1 (単位加工時間)

ジョブセットに含まれた注文書数およびロット数のばらつきを表現するために、注文書数は平均  $\mu_{order}$ 、分散  $\sigma_{order}^2$  の正規分布に従う乱数で生成し、ロット数は平均  $\mu_{lot}$ 、分散  $\sigma_{lot}^2$  の正規分布に従う乱数として生成する。ただし負の値および 0 は現実的でないため除外し、正の値のみを採用する。以上の手順で受注量とロットサイズにランダムネスを持つジョブセットを作る。

表 4: 注文書数およびロット数に関する確率分布パラメータ

パラメータ	説明
$\mu_{order}$	総注文書数の平均値
$\sigma_{order}^2$	総注文書数の分散
$\mu_{lot}$	各注文書に注文された製品ロット数の平均値
$\sigma_{lot}^2$	各注文書に注文された製品ロット数の分散

これにより、実際の生産現場に見られる受注量とロットサイズのランダムネスでデータとして生

---

成する。

ジョブセットに含まれる各注文書には、最大 6 種類の製品種別が含むことができる。製品種別は確率  $P = 0.7$  の二項分布に従い注文書に記載される。一つの注文書の製品種別に重複はない。注文書別の製品種別行列  $order$  を構成する。この注文書ごとに製品種別ごとの総注文件数  $n_s$  をロット数とする。1 ロットあたりの基板数は 10 枚とし、加工時間は定数で決まる工程以外はロット単位で決まる。各加工工程の処理時間は、 $n_s$  とあらかじめ機械別に与えた単位加工時間または基準加工時間パラメータで決定する。例えば、ドリル加工や電気検査など加工時間が枚数に比例すると考えられる工程では、 $n_s \times MT_m, m = \{1, 2, 3, 4, 5, 8, 10, 11, 12, 13\}$  で工程時間を設定し、エッチングや積層などバッチ処理的な工程では、製品枚数に依存しない基準時間  $MT_m, m = \{6, 7, 9\}$  を用いる。こうして得られた処理時間行列  $OT$  を得る。

$$OT_{s,q} = \begin{cases} n_s * MT_m & m \in \{1, 2, 3, 4, 5, 8, 10, 11, 12, 13\} \\ MT_m & m \in \{6, 7, 9\} \end{cases} \quad (5)$$

## 4.2 ジョブセットに含まれる注文データ生成

本実験のためにジョブセットと呼ぶ同一出荷日をもつ注文書の集合を 10000 セット作成した。その各ジョブセットは注文書を要素とし、注文書には製品種別とそのロット数が複数記載される。この記載内容をデータとして生成する手順をここでは述べる。本研究における注文データ生成では、注文書数分布のパラメータを  $\mu_{order} = 10, \sigma_{order}^2 = 2$ 、ロット数分布のパラメータを  $\mu_{lot} = 3, \sigma_{lot}^2 = 1$  として設定した。

パラメータ設定の平均や分散を用いて実際の生産現場に見られる受注量とロットサイズのランダムネスでデータとして生成する。具体的なデータ生成は Python で実装した。その手順を以下にまとめる。

**手順 1:** 亂数  $seed_{value}$  を設定し、注文書数およびロット数の正規分布パラメータ、ならびに機械別単位加工時間・基準加工時間パラメータを初期化する。

**手順 2:** 生成するジョブセット数を  $gen = 10000$  とし、 $gen = 0, \dots, gen - 1$  について手順 2~8 を

---

繰り返す.

**手順3:** 注文書数の候補として, 平均  $\mu_{order}$ , 分散  $\sigma_{order}^2$  の正規分布からサンプルサイズ  $order_{size}$  個の乱数系列を生成する. 生成された乱数のうち 0 以下の値を除外し, 残った候補の中から 1 つをランダムに選択して, 実際の注文書数  $order_{number}$  とする.

**手順4:** 同様に, 平均  $\mu_{lot}$ , 分散  $\sigma_{lot}^2$  の正規分布からロット数候補を生成し, 0 以下を除外した上でロット数の候補集合を得る.

**手順5:** 受注数  $order_{number}$  と製品種類数 6 に基づき, 注文書原始データ行列  $order$  (サイズ  $order_{number} \times count_{type}$ ) を初期化する. 各注文書  $O_i$  と製品種別  $J_s$  について, 確率  $P = 0.7$  で当該製品が注文に含まれるとし, 含まれる場合にはロット数候補集合から 1 つをランダムに選択して  $order_{i,s}$  に代入する.

**手順6:** 行列  $order$  の列方向の合計に, 基板 1 ロットあたり 10 枚を乗じることで, 各製品種別ごとの総基板枚数ベクトル  $product_{count}$  を算出する.

**手順7:** 各製品種別  $J_s$  および工程番号  $q$  について, 機械割当行列  $OM$  の要素  $OM_{s,q}$  を参照し, 処理時間行列  $OT_{s,q}$  を決定する. 枚数に比例する工程では  $product_{count}[s]$  と単位加工時間を乗じ, バッチ処理工程では基準加工時間をそのまま用いる.

**手順8:** 行列  $OM$  および  $OT$  を用いて, 行列  $JOBSET$  を作成する.  $JOBSET$  は行方向に製品種別 (製品 1 製品 6), 列方向に工程番号 (加工工程 1 加工工程 17) を取り, 各要素に「(機械番号, 加工時間)」の組を文字列として格納する. これにより,  $JOBSET$  の機械番号部分が  $OM$ , 加工時間部分が  $OT$  に対応する.

ただし, 同じジョブの作業について, 同じ機械で連続的な二つの加工工程を加工しないように, 作業の前作業と後作業のどちらの作業機械は一致しないと設定し, 次の条件を加えている.

$$\begin{cases} \text{before}\{m_{s,q}\} \neq m_{s,q} \\ \text{after}\{m_{s,q}\} \neq m_{s,q} \end{cases} \quad (6)$$

---

**手順9:** データの保存（分布種別，受注数分布パラメータ，ロット数分布パラメータ，使用機械数，シード値）に基づいて出力ディレクトリおよびファイル名を構成し，*JOBSET* を .csv 形式で保存する。

### 4.3 データ生成結果

先節で述べた通りに 10000 個のジョブセットデータを実際に生成した。最適化の実験ではこの 10000 のジョブセットから 120 個をランダムに選択した。本研究はこの 121 個のジョブセットについて 1 ジョブセットは 1 つの実験セットの単位として遺伝的アルゴリズムまたは深層強化学習を用いて最適スケジュール案を探査する。この 120 個のジョブセットを実験用ジョブセットと呼ぶ。

$$\overline{OT} = \frac{1}{N} \sum_{s=1}^6 \sum_{q=1}^{Q_s} OT_{s,q} \quad (7)$$

$$V = \frac{1}{N} \sum_{s=1}^6 \sum_{q=1}^{Q_s} (OT_{s,q} - \overline{OT})^2. \quad (8)$$

この実験用ジョブセットに含まれる 120 ジョブセットの加工工程の加工時間  $OT_{s,q}$  ( $OT_{s,q} > 0$ ) の平均加工時間  $\overline{OT}$  (式 7) と分散  $V$  (式 8) を計算し分散の大きさを基準として、三分位でジョブセットの特徴を含まれる注文書の多様性で小 (small)、中 (regular) と大 (large) のように 3 タイプに分類する。第 1 四分位点を  $Q_1$ 、第 3 四分位点を  $Q_3$  とすると、各ジョブセットは以下の 3 種類に分類される。

- **small:**  $V \leq Q_1$  （注文書内容の均一性が高く、多様性が小）
- **regular:**  $Q_1 < V \leq Q_3$  （注文書内容の多様性が中）
- **large:**  $V > Q_3$  （注文書内容のばらつきが大きく、多様性が大）

注文書の多様性が小 (small) とは、比較的注文書内容の均一性が高く、一方で大 (large) であれば、注文書の内容のばらつきが大きく、生産スケジュールを計算する際に、非常に短時間で加工がおわる単純な加工依頼から、ほぼ全加工機械の処理を必要とし、ロット数の変動も大きなものまで対象

に、様々な生産管理スケジュールを検討しなくてはならない。計算機実験での最適化の評価は3タイプの属性選択されたデータを次章の実験で利用する。

<b>id</b>	<b>タイプ</b>	<b>分散</b>						
1	regular	291969	41	large	411176	81	small	162249
2	regular	319267	42	small	138922	82	large	380666
3	regular	188773	43	small	182554	83	small	78159
4	regular	294718	44	regular	201636	84	regular	229246
5	small	110509	45	small	63946	85	regular	299946
6	regular	247450	46	large	597109	86	large	344939
7	regular	260110	47	regular	301418	87	large	370694
8	regular	324852	48	large	327616	88	large	364219
9	small	188229	49	regular	251280	89	small	130342
10	small	188605	50	small	150776	90	large	388925
11	small	152457	51	regular	245017	91	regular	258826
12	large	672144	52	large	343286	92	regular	257120
13	small	102620	53	regular	190370	93	large	392275
14	small	131369	54	small	185744	94	small	176455
15	small	171182	55	small	155211	95	large	432331
16	regular	272638	56	small	130460	96	regular	283359
17	regular	292862	57	large	339204	97	small	113377
18	regular	236119	58	large	365846	98	small	144882
19	small	182212	59	regular	270086	99	small	188580
20	large	391714	60	large	343896	100	large	379744
21	regular	209853	61	large	409258	101	regular	292252
22	regular	205243	62	regular	312287	102	regular	305207
23	large	366098	63	large	370204	103	large	626685
24	large	386418	64	small	139459	104	small	159088
25	large	405649	65	small	148883	105	large	356261
26	small	89952	66	regular	265198	106	large	380604
27	large	331674	67	large	569503	107	small	100954
28	regular	224601	68	regular	301313	108	large	397146
29	small	129098	69	small	106460	109	small	109580
30	large	394861	70	regular	297568	110	small	182488
31	large	592042	71	large	442688	111	large	376685
32	small	168738	72	regular	325593	112	small	137131
33	regular	253820	73	large	612936	113	small	119496
34	regular	299788	74	large	387869	114	small	163611
35	small	106248	75	small	120854	115	regular	295327
36	large	326180	76	large	331365	116	small	115524
37	regular	199462	77	large	484482	117	large	347403
38	regular	244236	78	regular	324833	118	large	352087
39	regular	210357	79	small	183299	119	large	420986
40	regular	210132	80	small	170160	120	regular	314926

図 2: データ分類

## 5. 遺伝的アルゴリズムによる解の改善

本研究では,JSSP に対して,GA を用いたスケジューリングの最適解探索手法を採用する. 加工順序が厳密に定まる問題なので,PPS(Precedence Preservative Crossover) という交差手法を採用する.

### 5.1 遺伝的アルゴリズムの考え方

本研究では,JSSP に対してジョブごとの工程順序を厳密に保持しつつ, 複数の製品に属する加工工程の加工順序を染色体として表現する遺伝的アルゴリズム (GA) を用い, 解の改善を行う.

まず, まとめた注文書の束であるジョブセットの各製品種別  $s$  の注文のジョブ  $J_s$  の加工工程列は, 機械割当行列  $OM$  および加工時間行列  $OT$  に基づく. 1 つのジョブは、「使用機械  $m_{s,q}$  と加工時間  $t_{s,q}$  の列からなる工程」として表現される. まず染色体は, 全ジョブの各工程を含む遺伝子列

$$\text{genes} = [J_s F_q, J_{s'} F_{q'}, \dots]$$

を定義する。各遺伝子は「ジョブ  $J_s$  の第  $q$  工程」を表す. スケジュールの手順について, 遺伝子列に従って工程を順に配置し, 同一製品ジョブ内の工程順序制約および各機械が同時に 1 工程しか処理できないという制約を満たすように処理開始時刻と終了時刻を更新し, スケジュール全体の総加工時間を算出する.

まず遺伝的アルゴリズムのスタート時の初期解として, 各製品ジョブ内の加工工程順序を保したランダムな遺伝子列に導入する方法で生成する. 具体的には, 各ジョブについて「次に処理すべき工程インデックス」を管理し, 未処理工程を有するジョブの集合からランダムに 1 ジョブを選択し, そのジョブの次の加工工程を遺伝子列の末尾に追加する操作を繰り返す. この手続きを全ジョブの全加工工程が配置されるまで継続することで, すべての工程を一度ずつ含み, かつ製品ジョブ内の加工工程順序が必ず守られた初期解が得られる.

GA の適応度評価は, 各染色体が表現するスケジュールの品質, ここでは JSSP における代表的な性能指標である総加工時間を評価基準として採用する. 既存のスケジュール案から次のスケジュール案を作成するために, 各染色体が示す遺伝子列をスケジュールへとデコードし, 製品ジョブ内の工程順序制約および機械資源制約を満たす実行計画を構成した上で, 対応する総加工時間を算出す

---

ることで、よりよいスケジュール案の総加工時間を比較し採用する。

適応度評価では、各染色体に対してスケジュールを配置し、対応する総加工時間を適応度として計算する。本研究では総加工時間の最小化を目的とするため、適応度は値が小さいほど良好な解であることを意味する。計算された総加工時間は染色体オブジェクト内に保存され、同一染色体に対する重複総加工時間計算を避けることで、計算効率の向上を図っている。

親個体の選択にはトーナメント選択を採用する。許容解集団からランダムに複数個体（本研究では 5 個体）を抽出し、その中で総加工時間が最小の個体を勝者として選択する操作を 2 回行うことで、二つの親染色体を決定する。この選択方式により、適応度の高い個体が次世代に残りやすくなる一方で、一定のランダム性が維持され、探索の多様性が確保される。

GA で二つの親から特徴を継承し子個体を生成するためには交叉操作が必要です。交叉操作には、PPS(Precedence Preservative Crossover) と呼ばれる順序保持型の交叉手法を用いる。この交差手法の紹介は次章で述べる。生成後は染色体修復処理を行う、加工工程の欠落や重複が生じないよう、解の有効性を保証する。

また、本研究の GA ではエリート保存戦略を採用している。各世代において総加工時間が最小となる個体から、あらかじめ定めた個数（本研究では 2 個体）を次世代集団へそのまま引き継ぐことで、進化の過程で得られた最良解が失われることを防止する。残りの個体は、選択・交叉操作によって生成された子個体で構成され、所定の最大ジェネレーション世代数（本実装では 200 世代）に達するまで進化計算を繰り返す。最終的に全ジェネレーション世代を通じて得られた最良染色体を GA の解として採用し、対応するスケジュールについてガントチャートと各ジェネレーション世代にもっとも短い総加工時間 (*makespan*) の折れ線グラフを描画し、視覚的に評価の適切さを確認できる。

## 5.2 PPS 交差法の考え方

PPS 交差法は、JSSP 問題などの順序制約を伴う組合せ最適化問題において有効な交差手法の一つである。本手法は親個体の遺伝子列情報を交差する際に、工程間の優先順序を保持しつつ、新たな子個体を生成することを目的としている。

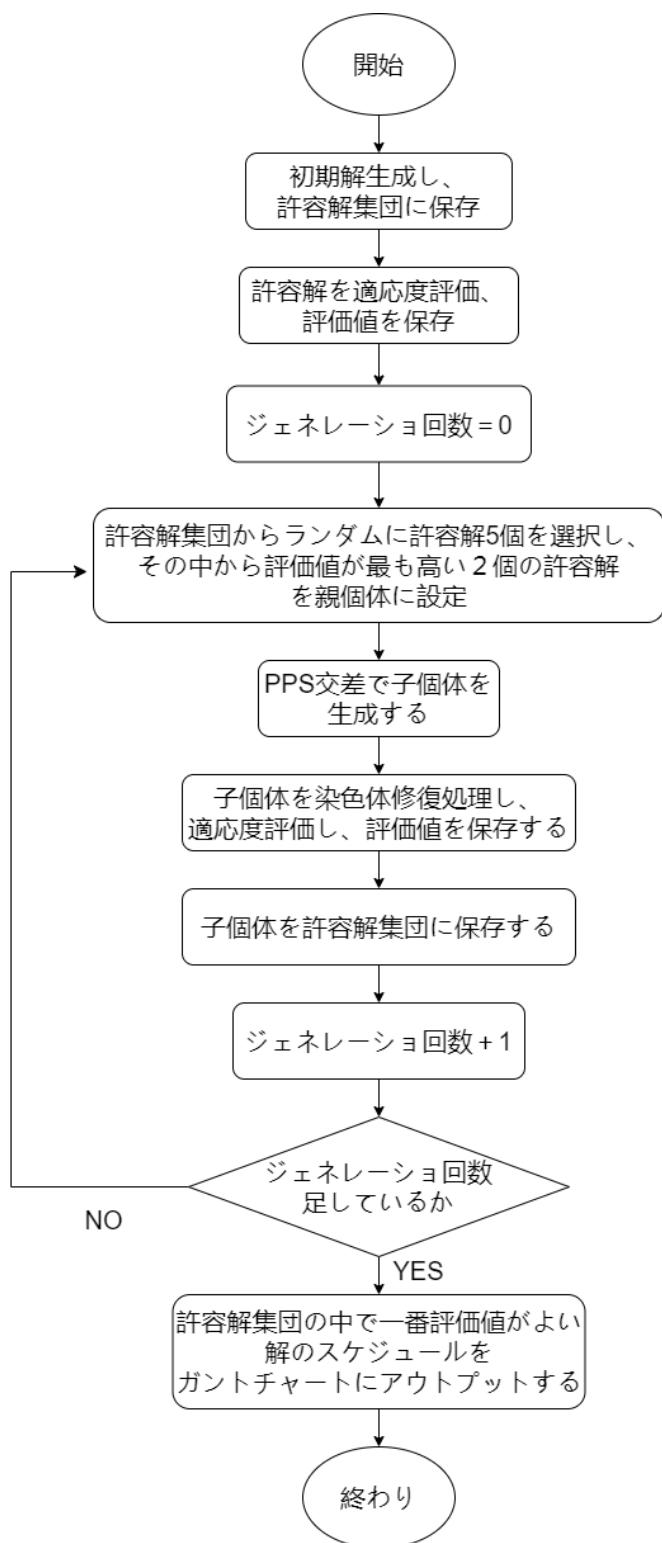


図 3: 遺伝的アルゴリズムフローチャート

---

PPS の交差手順は次の通りである.

1. 2つの親染色体 (Parent1, Parent2) を許容解集団から選択する.
2. Parent1 と Parent2 から操作配列を主に利用する.
3. 子個体の染色体を初期化する.
4. Parent1 と Parent2 をランダムに選択する. 選択された親個体の頭から, 選択されない操作を子個体操作配列の最後に挿入する.
5. すべての操作が配置されるまで 4 を繰り返す.

記号 :

- $J = \{J_1, J_2, \dots, J_s\}$  : 製品種類の集合
- 各製品  $J_s$  は  $q$  個の加工操作を持ち,  $F_q = \{J_s F_1, J_s F_2, \dots, J_s F_q\}$  とする.
- 親個体の操作列 :  $P_1 = [o_1^1, o_2^1, \dots, o_\theta^1]$ ,  $P_2 = [o_1^2, o_2^2, \dots, o_\theta^2]$
- 親個体の頭からまだ子個体に配置していない操作 :  $P_1 o \in [o_1^1, o_2^1, \dots, o_\theta^1], P_1 o \notin C$  、  $P_2 o \in [o_1^2, o_2^2, \dots, o_\theta^2], P_2 o \notin C$
- 子個体  $C$  を空列として初期化 :  $C \leftarrow []$

**step1.**  $i \leftarrow 1$

**step2.** while  $|C| < \theta$  do:

**step2-1.**  $Po = random\{P_1 o, P_2 o\}$

**step2-2.**  $C \leftarrow Po$

**step2-3.**  $i \leftarrow i + 1$

**step3.** return  $C$

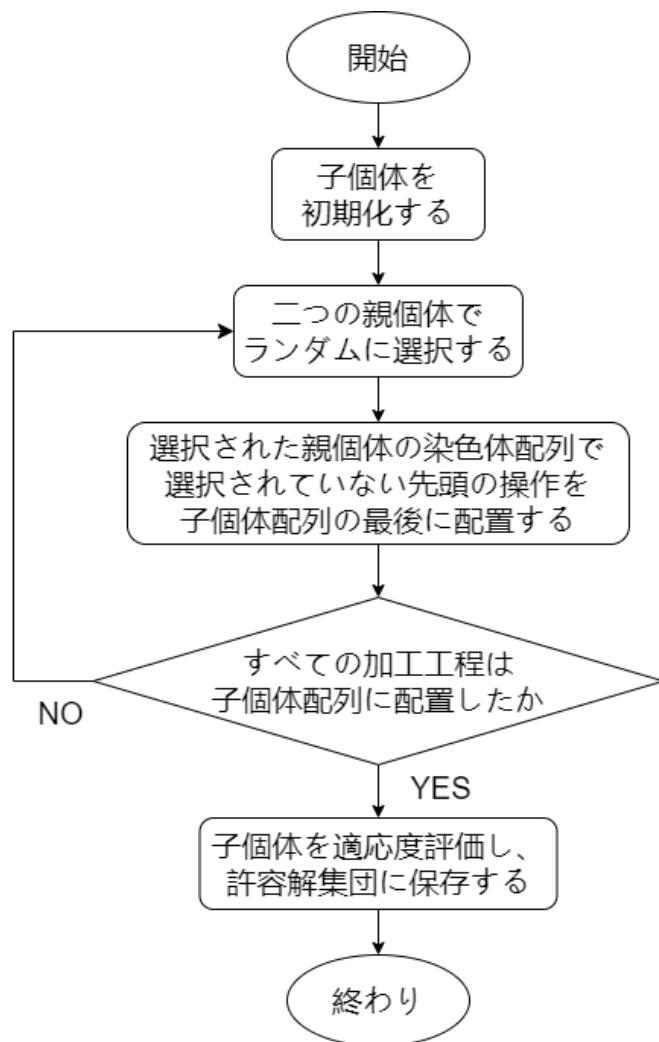


図 4: PPS フローチャート

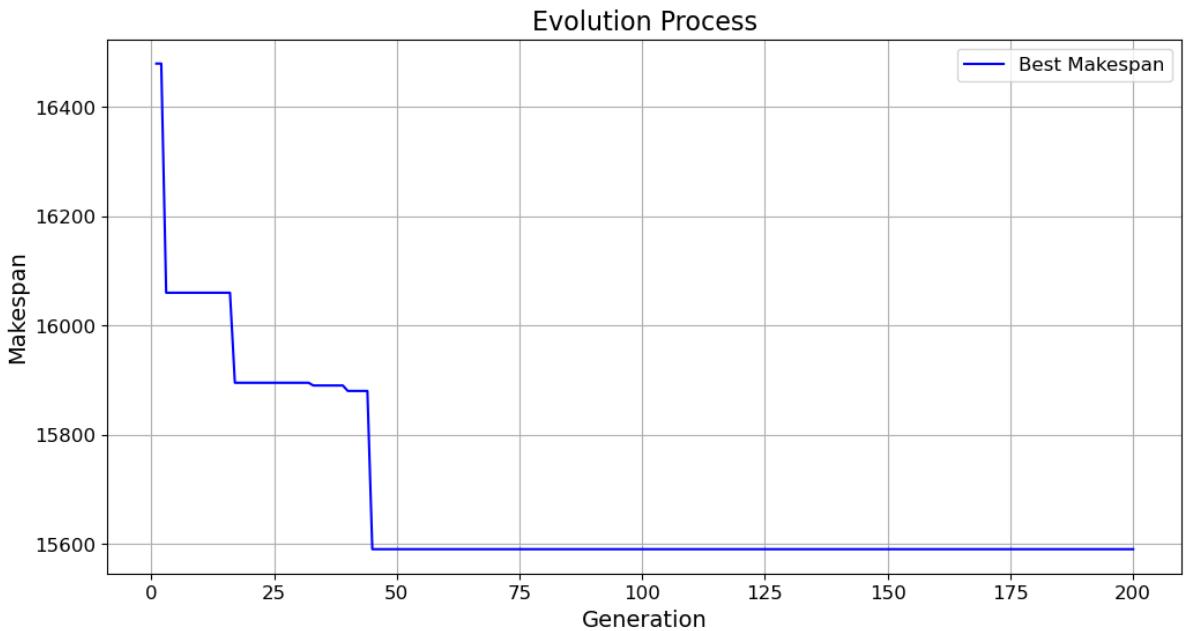
### 5.3 GA の実験結果

以上の手順で実験用ジョブセットにある 120 個データをスケジュールして、最後の許容解 *makespan* とガントチャートを結果として出力した。

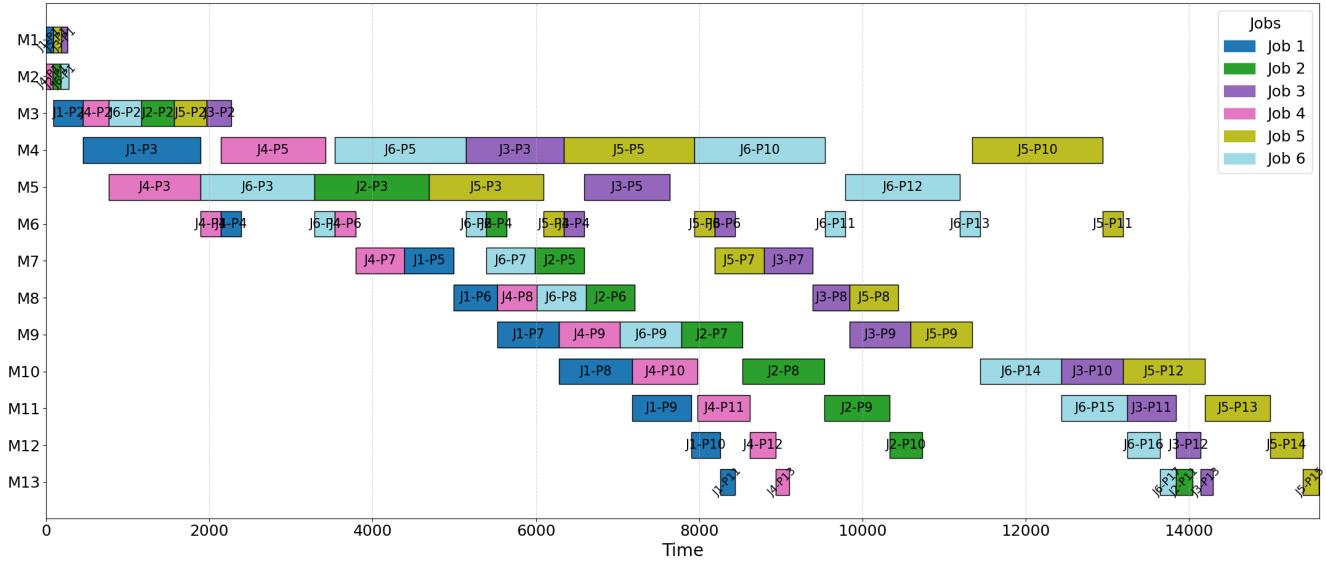
<i>id</i>	タイプ	GA makespan						
1	regular	18665	41	large	21210	81	small	13035
2	regular	19085	42	small	12575	82	large	19930
3	regular	15300	43	small	14135	83	small	9945
4	regular	18510	44	regular	16365	84	regular	16290
5	small	12740	45	small	8585	85	regular	17830
6	regular	16660	46	large	24050	86	large	20235
7	regular	18270	47	regular	18410	87	large	19940
8	regular	19485	48	large	18945	88	large	20230
9	small	14860	49	regular	17385	89	small	13080
10	small	14385	50	small	13395	90	large	20515
11	small	14030	51	regular	15380	91	regular	16700
12	large	26620	52	large	18820	92	regular	17335
13	small	11310	53	regular	14660	93	large	20995
14	small	12840	54	small	14160	94	small	14500
15	small	14375	55	small	14340	95	large	21030
16	regular	15950	56	small	13630	96	regular	17845
17	regular	17505	57	large	18840	97	small	12330
18	regular	16400	58	large	20120	98	small	12825
19	small	15100	59	regular	16180	99	small	14000
20	large	20435	60	large	19310	100	large	20535
21	regular	15310	61	large	21265	101	regular	16120
22	regular	15410	62	regular	18925	102	regular	18625
23	large	19655	63	large	21065	103	large	25225
24	large	19955	64	small	13080	104	small	14430
25	large	20820	65	small	13655	105	large	19000
26	small	9865	66	regular	17690	106	large	20255
27	large	18330	67	large	24375	107	small	10575
28	regular	15410	68	regular	18795	108	large	20685
29	small	12755	69	small	11475	109	small	11805
30	large	21430	70	regular	19055	110	small	15000
31	large	24855	71	large	21900	111	large	20095
32	small	13660	72	regular	19990	112	small	13415
33	regular	17900	73	large	25270	113	small	12455
34	regular	18780	74	large	20270	114	small	14520
35	small	11995	75	small	13005	115	regular	19000
36	large	19640	76	large	18955	116	small	12460
37	regular	15320	77	large	21975	117	large	18450
38	regular	16950	78	regular	18940	118	large	19650
39	regular	15685	79	small	14875	119	large	21220
40	regular	16565	80	small	14170	120	regular	18490

図 5: GA が output した 120 ジョブセット (注文タイプ別) の *makespan*

本研究を背景としての問題設定およびジョブセットの規模でしたら、ジェネレーション終えて許容解を出せるまで概ね 17 秒左右かかる。以下は注文タイプにより、三つ種類の中でランダムに選択された注文のジェネレーションの最短総加工時間折れ線グラフとガントチャート。

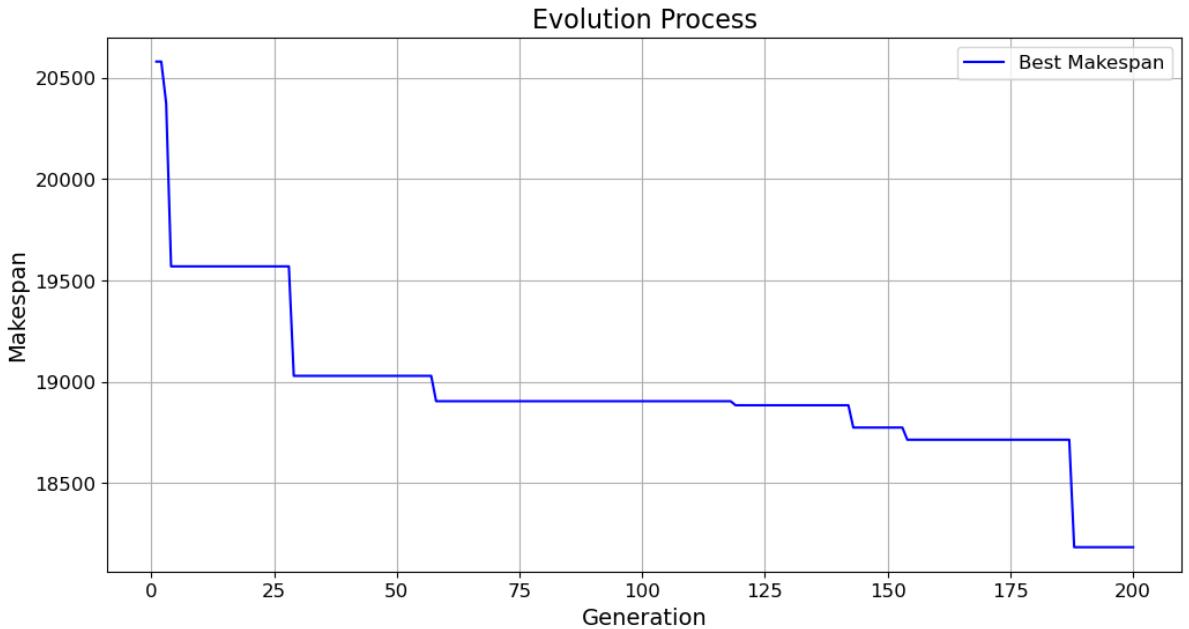


(a) ジョブセット 19(小) のジェネレーション折れ線グラフ  
GA-based JSSP Gantt Chart (Makespan = 15590)

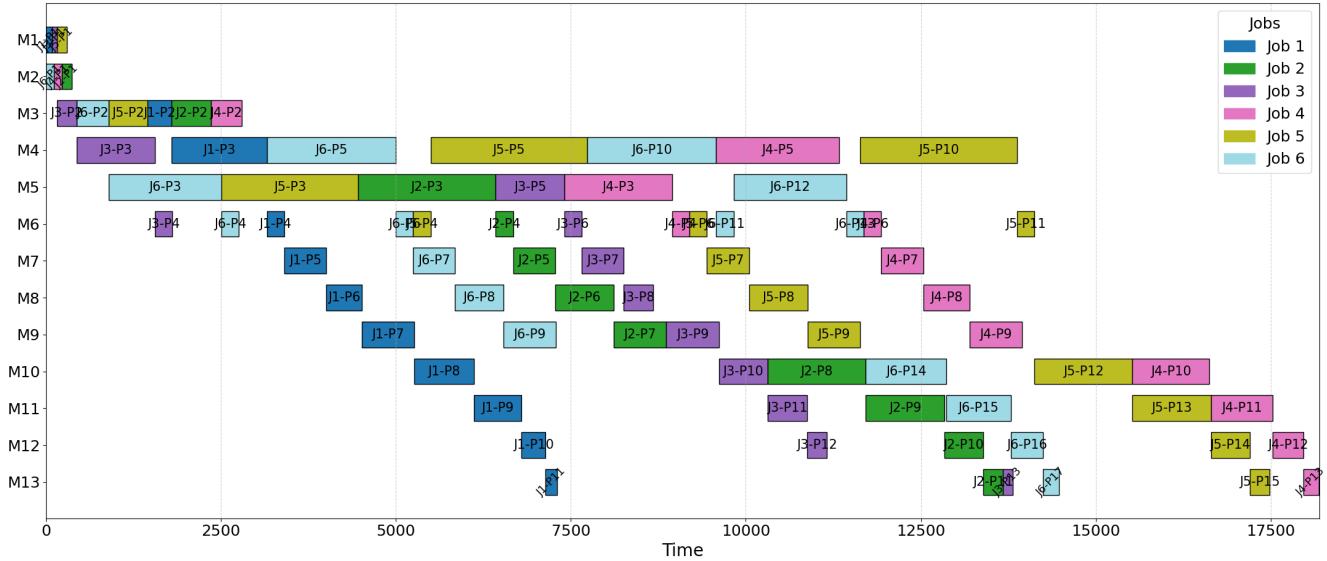


(b) ジョブセット 19(小) の許容解ガントチャート

図 6: ジョブセット 19(小) の実験結果

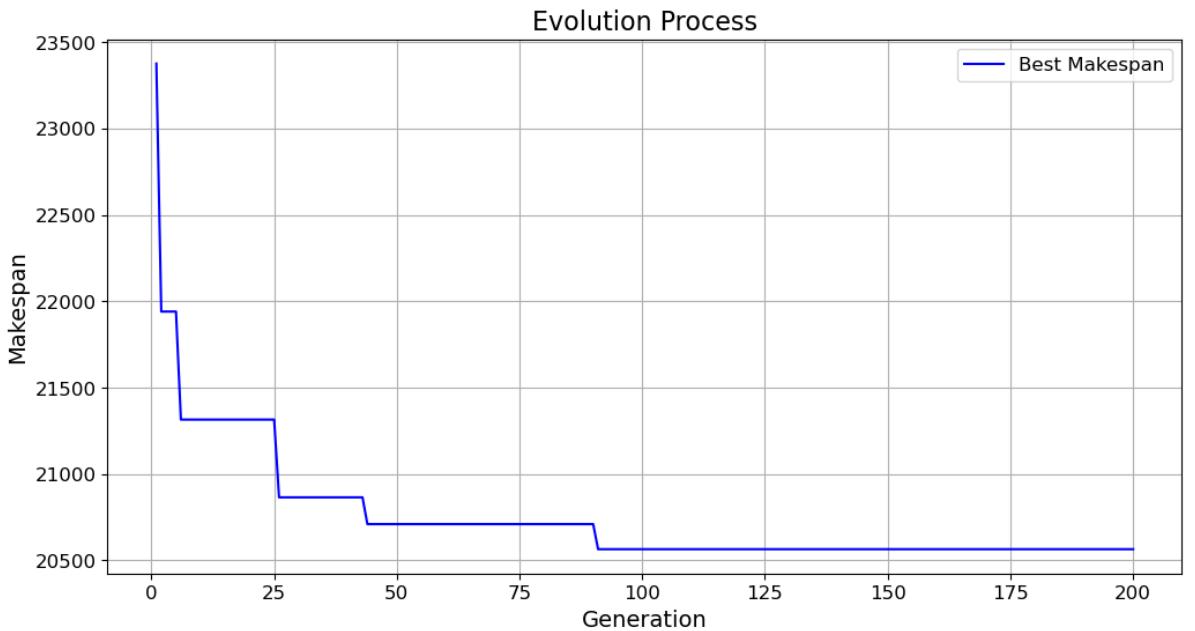


(a) ジョブセット 68(中) のジェネレーション折れ線グラフ  
GA-based JSSP Gantt Chart (Makespan = 18185)

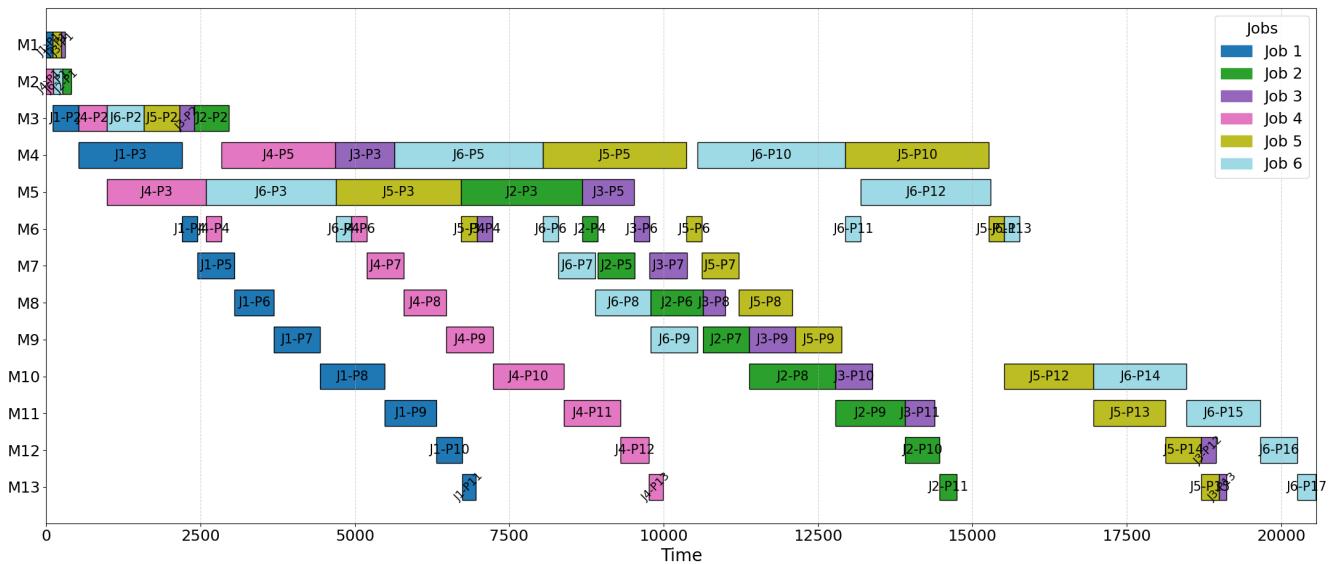


(b) ジョブセット 68(中) の許容解ガントチャート

図 7: ジョブセット 68(中) の実験結果



(a) ジョブセット 93(大) のジェネレーション折れ線グラフ  
GA-based JSSP Gantt Chart (Makespan = 20565)



(b) ジョブセット 93(大) の許容解ガントチャート

図 8: ジョブセット 93(大) の実験結果

---

GA の実験結果を見て, ジェネレーション回数は 50 回以下で, 200 回の計算終了時と変わらない最適解を得ていることがわかる. 収束まで最も計算回数の多いケースでも 200 回以下で良いスケジューリング案に到達している. ガントチャートからは, 加工工程のスケジューリングの可視化結果から, 機械の遊休時間がほとんど見られないスケジュール案が出力できることも視認できる.

また, GA によるスケジュール最適案は初期解と比べて, 10% から 20% まで *makespan* を改善できる. JSSP は典型的な NP 問題であるので, 問題により厳密解に至るまで総当たり法が必要となる. それを踏まえれば GA は JSSP のスケジュール最適化で実効性の高い計算時間で最適解をだせることがわかる.

次節から, 深層強化学習による JSSP のスケジュール最適化性能について, 図 5 に示す, GA で得られたデータをベースとして取り扱う. また, GA は確率的探索に基づくメタヒューリстиクスであり, 初期個体の生成や交叉の影響により, 同一のジョブセットに対しても毎回必ずしも同一のスケジュール解が得られるとは限らない. そのため, 図 5 に示す *makespan* は, 各ジョブセットについて GA を一度実行した際に得られた最終解の結果である. 以降の節において GA と DQN の性能比較を行う際には, 同一のジョブセットに対して再度 GA を適用した場合, 図 5 に示した値と完全には一致せず, 一定のばらつき (誤差) を含む可能性がある点に留意する必要がある.

---

## 6. 深層強化学習による解の改善

深層強化学習 (Deep Reinforcement Learning, Deep RL) とは、強化学習 (Reinforcement Learning, RL) と深層学習を組み合わせた機械学習の一分野である。RL は意思決定問題を扱う手法として広く用いられており、一般にマルコフ決定過程 (Markov Decision Process、MDP) として定式化される。RL はエージェントと環境から構成され、エージェントは環境との相互作用を通じて、全体として最適なポリシーを探索する。

エージェントはまず環境の状態を観測し、現在保持している方策に基づいて行動候補の中から行動を選択する。その行動により環境の状態が変化し、エージェントは新たな環境において再び行動を選択する。この過程において環境から報酬が与えられ、エージェントは将来にわたる報酬を定期的に評価することで、ポリシーの更新および改善を行う。

RL と Deep RL の主な違いは、行動選択の際にニューラルネットワークを用いる点にある。Deep RL では、ニューラルネットワークによって状態と行動の関係を近似することで、行動候補の中から適切な行動を選択する。これにより、複雑なモデルを比較的簡潔な形式で表現できるとともに、意思決定に関与する多様な要因を柔軟に考慮することが可能となる。

さらに、Deep RLにおいては、エージェントが試行錯誤を通じて方策を学習・改良することで最適ポリシーを獲得し、動的に変化する環境においても、その最適ポリシーに基づいた意思決定が可能となる。

### 6.1 深層強化学習の考え方

強化学習は環境との相互作用を通じて逐次的に意思決定を行う枠組みであり、一般にマルコフ決定過程 (MDP) として表現される。MDP は、状態集合 (環境)  $S$ 、行動集合  $A$ 、報酬関数  $R$ 、および割引率  $\gamma$  から構成される。

本研究では、価値関数に基づく強化学習手法である Deep Q-Network (DQN) を用いる。DQN では、各状態  $s_t$  における行動  $a_t$  の価値を表す行動価値関数  $Q(s_t, a_t)$  をニューラルネットワークにより近似し、各状態における行動選択を  $Q$  値に基づいて行う。図 9 に DQN の仕組みとして手法の概要を示す。また、表 5 に使用する記号をまとめる。

---

エージェント時刻  $t$  において環境の状態  $s_t \in S$  を観測し,  $Q(s_t, a)$  が最大となる行動を選択することで, 将来にわたる累積報酬の最大化を目指す. 学習過程においては,探索とすでに学習した特徴の活用のバランスを取るため,  $\varepsilon$ -greedy 法を用いて確率的にランダム行動を選択する.

時刻  $t$  において,  $[0, 1]$  の一様分布から乱数  $u$  を生成し, 行動  $a_t$  は次式により決定される:

$$a_t = \begin{cases} \text{random}(A_t) & \text{if } u < \varepsilon, \\ \arg \max_{a_t \in A_t} Q(s_t, a_t; \theta) & \text{if } u \geq \varepsilon. \end{cases} \quad (9)$$

なお, 本研究では学習の進行に伴い  $\varepsilon$  を段階的に減少させることで, 初期段階では探索を重視し, 学習が進むにつれて活用を優先する設計としている.

本研究で扱う JSSP 環境は, 加工工程の開始および完了といったイベントによって状態が更新されるイベント駆動型の環境として構成されている.

DQN における 1 step とは, 時刻  $t$  においてエージェントが状態  $s_t$  を観測し, 行動  $a_t$  を選択した後, 環境が次状態  $s_{t+1}$  と報酬  $r_t$  を返すまでの連続的な意思決定単位を指す.

行動  $a_t$  には,

- 実行可能な作業工程を選択してスケジューリングを行う行動
- 次のイベント発生まで待機する行動

の両方が含まれており, 各 step において環境状態が更新される.

そして, episode とは, 一つのジョブセットに対して, 初期状態から開始し, すべての作業工程が完了するまでのことを指す. すなわち, 1 episode は一つのジョブセットに対するすべての加工プロセスが配置された, 完全なスケジューリング過程に対応する.

本実装では, メインネットワークとターゲットネットワーク, 二つのネットワークを用いる. メインネットワークは, 状態  $s_t$  を入力として各行動の  $Q$  値を出力し,  $\varepsilon$ -greedy 法により実行行動  $a_t$  を決定するために用いられる. また, 経験再生でサンプルされた  $(s_t, a_t, r_t, s_{t+1})$  に対して, メインネットワーク出力から  $Q(s_t, a_t)$  を算出し, 誤差逆伝播によりメインネットワークのパラメータ更新を行う.

---

経験再生は DQN の学習においては、連続した経験データ間の相関を減り、学習の安定性とデータ効率を向上させるため導入する。

各 step において得られる経験  $(s_t, a_t, r_t, s_{t+1})$  は、リプレイバッファ (replay buffer、または replay memory pool にと呼ばれる) に保存される。学習時には、このバッファからランダムにミニバッチを抽出し、損失関数に基づいてメインネットワークの更新を行う。この手法により、

- 学習データの相関を低減できる
- 過去の重要な経験を繰り返し利用できる
- 学習の発散を抑制できる

といった効果が得られる。

また本研究では、一定数以上の経験  $(s_t, a_t, r_t, s_{t+1})$  が蓄積された後に経験再生による学習を開始することで、初期段階における不安定な更新を回避している。

ターゲットネットワークは、将来報酬に相当する非リアルタイムな価値成分の推定に用いられる。具体的には、次状態  $s_{t+1}$  に対してターゲットネットワークから  $\max_{a'} Q(s_{t+1}, a'; \theta^-)$  を計算し、

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) \quad (10)$$

として目標値  $y_t$  を構成する。これにより、即時に得られる報酬  $r_t$ （リアルタイムな報酬成分）と、将来の累積報酬に対応する価値（非リアルタイムな価値成分）を分離して扱うことができる。

さらに、本実装では学習の安定化のため、一定ステップ間隔ごとにメインネットワークのパラメータをターゲットネットワークへコピーするハード更新を行う。具体的には、現時点ステップ数が設定していた更新ステップの倍数となるたびに  $\theta^- \leftarrow \theta$  として同期する。また、固定テストデータ集合に対する定期評価では、評価値のばらつきを抑える目的とモデルの性能評価からターゲットネットワークを用いて計算を行う。

メインネットワークのパラメータ  $\theta$  の更新には、確率的勾配降下法に基づく最適化手法である Adam オプティマイザを用いる。学習率を  $\alpha$  とすると、各学習ステップにおけるパラメータ更新は、損失関数  $L_t(\theta)$  の勾配に基づいて行われる。以上の処理により、各ステップにおいてメイン

---

ネットワークのパラメータ更新が行われる。

DQN における学習は、メインネットワークが出力する行動価値  $Q(s_t, a_t; \theta)$  と、ターゲットネットワークを用いて構成される目標値  $y_t$  との差を最小化することを目的として行われる。

損失関数として平均二乗誤差 (Mean Squared Error, MSE) を採用し、時刻  $t$  における損失  $L_t(\theta)$  を次式で定義する：

$$L_t(\theta) = \mathbb{E} [(y_t - Q(s_t, a_t; \theta))^2]. \quad (11)$$

ここで、目標値  $y_t$  はターゲットネットワークを用いて式 10 として計算される。

表 5: 強化学習および DQN に用いる記号の定義

記号	説明
$S$	状態集合（環境が取り得る全状態の集合）
$A$	行動集合（エージェントが選択可能な行動の集合）
$s_t$	時刻 $t$ における環境の状態
$a_t$	時刻 $t$ において選択される行動
$A_t$	状態 $s_t$ における実行可能行動集合
$r_t$	時刻 $t$ に得られる即時報酬
$R$	報酬関数
$\alpha$	学習率
$\gamma$	将来報酬の重み付け係数
$Q(s, a)$	状態 $s$ において行動 $a$ を選択した際の行動価値
$Q(s, a; \theta)$	メインネットワークにより近似される行動価値関数
$Q(s, a; \theta^-)$	ターゲットネットワークにより近似される行動価値関数
$\theta$	メインネットワークのパラメータ
$\theta^-$	ターゲットネットワークのパラメータ
$y_t$	学習における目標値
$\varepsilon$	$\varepsilon$ -greedy 法における探索率
$a'$	次状態における行動候補
$L_t(\theta)$	時刻 $t$ における損失

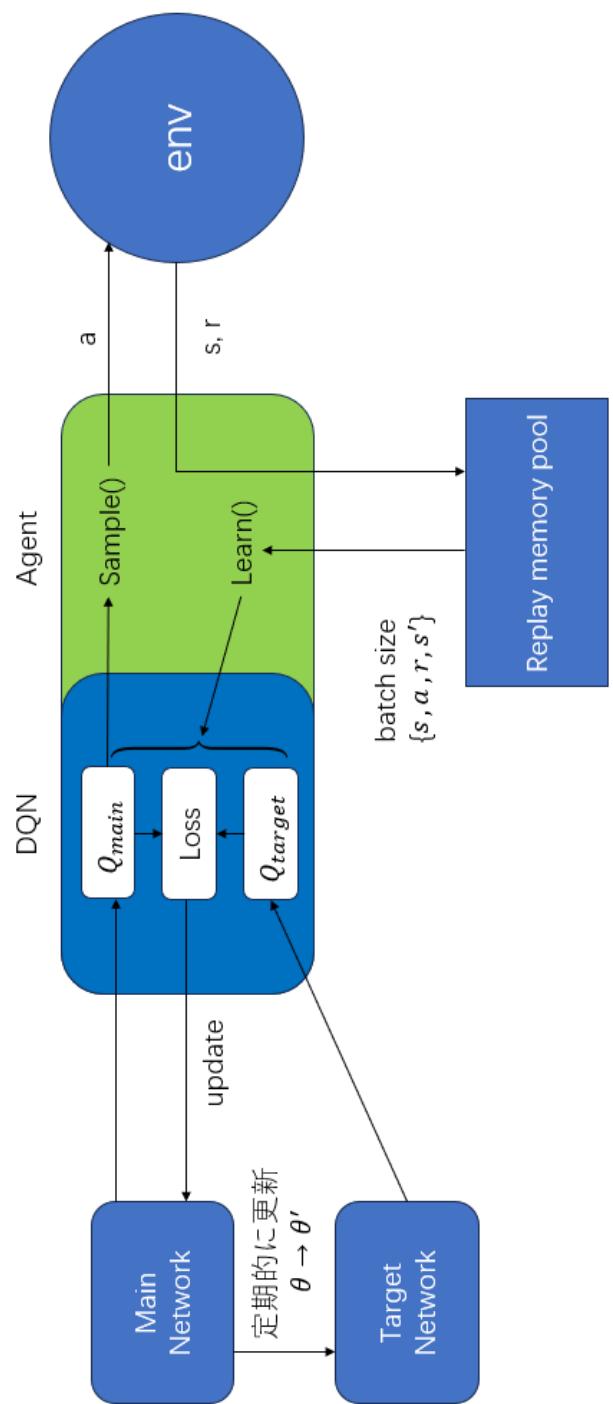


図 9: DQN の仕組み

## 6.2 DQN を JSSP に適用する理由

前章では, GA を用いて JSSP に対するスケジューリング最適化を行い, 比較的高品質な許容解を得られることを示した. GA は問題構造に依存せず適用可能であり, 事前学習を必要としない点で汎用性の高い手法である. 一方で, 各ジョブセットに対して毎回多数の世代にわたる進化計算を実行する必要があり, 1 回のスケジューリングに要する計算時間は問題規模に応じて増大する.

これに対し, 本研究で採用する DQN は, 環境との相互作用を通じてスケジューリングポリシーを Q-value によって学習する手法である. DQN では, 学習段階において多数のジョブセットを用いてモデルを訓練する必要があるものの, 一度学習が完了した後は, 各ジョブセットに対するスケジューリングを逐次的な推論処理として高速に実行できる. そのため, 同一規模のジョブセットに対する 1 回の完全スケジューリングに要する時間は, GA と比較して大幅に短縮される.

特に生産現場においては, 機械故障や新規受注の追加などによりスケジュールを頻繁に再計算する必要が生じる. このような状況では, 每回最初から探索を行う GA よりも, 学習済みモデルに基づいて即時に意思決定を行える DQN の方が適していると考えられる.

以上より, GA は静的環境における高精度なスケジューリング最適化に有効である一方, DQN は学習コストを伴うものの, 単一ジョブセットに対する計算時間が短く, 動的環境やリアルタイム性が要求される状況において優位性を有する. 本研究では, これらの特性を踏まえ, GA と DQN の両手法を比較対象として位置付ける.

## 6.3 報酬関数の構成

本研究では DQN によりジョブショップスケジューリングを逐次意思決定として解くため, 各 step における即時報酬  $r_t$  を設計する. 報酬設計は学習の収束性, アクション選択方策の偏りまたは訓練済みモデルの性能に強く影響するため,

- 複数指標を組み合わせた複合型報酬関数
- $makespan$  を直接最適化する  $makespan$  中心型報酬関数

の 2 種類を実装し, 比較した.

### 6.3.1 複合型報酬関数

複合型報酬では、スケジューリングの良さを单一指標に限定せず、「makespan の短縮」「各ジョブの加工プロセスの推進」「機械負荷の同一化」「待ち時間の抑制」など、複数の観点を同時に評価することで学習を安定化またはモデル性能の向上させることを狙う。

時刻  $t$  における報酬  $r_t$  は、以下の重み付き和として定義する：

$$r_t = r_t^{\text{time}} + r_t^{\text{prog}} + r_t^{\text{bal}} + r_t^{\text{comp}} + r_t^{\text{opp}} + r_t^{\text{util}} + r_t^{\text{cp}} + r_t^{\text{bn}} + r_t^{\text{final}}. \quad (12)$$

各項は以下の通りである（実装では係数はハイパーパラメータとして設定する）：

- **加工時間ペナルティ**: 選択した第  $s$  種別製品の第  $q$  番目の加工工程の加工時間を  $t_{s,q}$  とすると、

$$r_t^{\text{time}} = -w_{\text{time}} t_{s,q}. \quad (13)$$

長い工程の選択を抑制し、局所的な時間増加を抑える目的で導入する。

- **加工工程推進報酬**: 全工程数を  $N_{\text{ops}}$ 、当 step の完了工程数を  $N_{\text{step}}^f$  とし、進捗率を  $g_{\text{step}} = N_{\text{step}}^f / N_{\text{ops}}$  と定義する。

$$r_t^{\text{prog}} = w_{\text{prog}} (g_{\text{step}+1} - g_{\text{step}}). \quad (14)$$

1 step の意思決定がイベント駆動により「工程完了」という推進することを明示するための報酬である。

- **機械負荷平準化**: 機械  $M_m$  の累積使用時間を  $t_m^{\text{sum}}$  とし、

$$r_t^{\text{bal}} = -w_{\text{bal}} \text{std} (\{t_m^{\text{sum}}\}). \quad (15)$$

特定機械へのボトルネック化を抑える目的で導入する。

- 
- **ジョブスケジュール完了報酬**: あるジョブの最終工程が完了した場合に定数報酬を与える：

$$r_t^{comp} = \begin{cases} w_{comp} & (\text{ジョブ } s \text{ が完了}) \\ 0 & (\text{それ以外}) \end{cases}. \quad (16)$$

- **機会損失ペナルティ（最短工程の未選択）**：実行可能行動集合を  $A_t$  とし、その中で最短加工時間を  $t_{min}^A$  とする。選択した第  $s$  種別製品の第  $q$  番目の加工工程の加工時間  $t_{s,q}$  が  $t_{min}^A$  より大きい場合、

$$r_t^{opp} = -w_{opp}(t_{s,q} - t_{min}^A). \quad (17)$$

「すぐ終わる作業」の優先を促す報酬である。

- **アイドル抑制（空き機械の即時利用）**：空き機械が存在するにもかかわらず開始が遅れる状況を抑えるため、機械が隙間がないように稼働させた場合は小さな正な報酬、逆に機械加工待ちが増える場合は軽微な負報酬を与える。
- **残作業最大ジョブ優先**：残り作業量の最も大きいジョブを優先した場合に小さな正な報酬を与える。
- **ボトルネック回避**：現時点で機械が残した加工工程が一番多いのをボトルネックとみなし、その機械をさらに悪化させる選択には軽微な負な報酬を与える一方、非ボトルネック機械を活用する選択には正報酬を与える。
- **エピソード終端の効率報酬**：すべてのジョブの加工工程が完了し、エピソードが終了した時点で、当該スケジューリング案の総加工時間  $makespan$  を  $C_{max}$  とする。また、全加工工程の加工時間総和を

$$T^{sum} = \sum_s \sum_q t_{s,q} \quad (18)$$

と定義し、加工機械台数を  $|M|$  とすると、理論的な下界として

$$C^{LB} = \frac{T^{sum}}{|M|} \quad (19)$$

---

を用いる。

エピソード終端時の報酬は、実際の  $makespan$  が理論下界にどの程度近いかを評価する指標として、以下のように定義する：

$$r_t^{final} = \begin{cases} w_{final} \frac{C^{LB}}{C_{\max}} & (\text{エピソード終端時}) \\ 0 & (\text{それ以外}) \end{cases}. \quad (20)$$

この終端報酬により、エージェントは各 step における局所的な判断だけでなく、エピソード全体として  $makespan$  を短縮する方策を学習することが期待される。

一方で、複合型報酬は各項が相互に干渉しやすく、重みの調整によっては「局所的に見えた良い行動」を過度に促進してしまい、目的である  $C_{\max}$  最小化と整合しない方策に収束する可能性がある。本研究の実験でも、複合型報酬は学習の安定性は得られる場合がある一方、最終的な  $makespan$  改善が限定的となる傾向が確認された。

### 6.3.2 $makespan$ 中心型報酬関数

次に、目的関数と報酬の整合性を高めるため、 $makespan$  の増分に基づく中心型報酬関数を設計する。step における  $makespan$  (推測完了時刻の最大値) を  $C_{\max}(t)$  とすると、

$$r_t^{\text{main}} = -\Delta C_{\max}(t) = -(C_{\max}(t+1) - C_{\max}(t)). \quad (21)$$

すなわち、意思決定によって  $C_{\max}$  が増えるほど負の報酬となり、 $C_{\max}$  の増加を抑える行動を学習する。

さらに、実装上はプロセス推進の停滞（無意味な待機や不適切な選択可能な加工プロセススキップ）を抑制するため、以下の補助項を加える：

$$r_t = r_t^{\text{main}} + r_t^{\text{imm}} + r_t^{\text{pen}} \quad (22)$$

- **即時着手ボーナス**  $r_t^{\text{imm}}$ : 「最も早く待機している機械」に対して直ちに工程を開始できた場合に小さな正報酬を与える。これは  $C_{\max}$  だけでは区別しにくい局所的な停滞を抑えるため

---

である。

- 待機・プロセススキップに関連ペナルティ  $r_t^{\text{pen}}$ : 実行可能工程が存在するにもかかわらず待機する、または不適切なスキップを繰り返すなど、スケジュール生成に推進しない行動に負報酬を与える。

この *makespan* 中心型報酬関数は目的関数  $C_{\max}$  と直接かかわっているため、報酬設計によるバイアスが小さく、本研究の実験では複合型より *makespan* を安定的に改善できる傾向が確認された。

## 6.4 モデル訓練結果

本章では、DQN のモデル訓練結果について評価を行う。具体的には、複合型報酬関数および *makespan* 中心型報酬関数のそれぞれに対して学習を行い、得られた訓練済みモデル（すなわちニューラルネットワークのパラメータ）を用いて、実際の加工順序行列  $OM$  および加工時間行列  $OT$  に対するスケジューリングを実行し、生成されたスケジュールの性能を評価する。

深層強化学習においては、モデルの性能がハイパーパラメータの設定に大きく依存することが知られている。特に学習の安定性、収束速度、および最終的に得られる解の品質は、学習率や割引率、探索率などのパラメータ設定によって大きく左右される。そのため、本研究では複数のハイパーパラメータの組み合わせを試行し、学習挙動およびスケジューリング結果の両面から性能評価を行った。複合型報酬関数および *makespan* 中心型報酬関数のいずれにおいても、共通して調整対象とした主なハイパーパラメータを表 6 に示す。

表 6: モデル訓練で調整するハイパーパラメータ

記号	説明
EPISODES	学習エピソード回数
$\alpha$	学習率
$\gamma$	将来報酬の重み付け係数
$\varepsilon$	$\varepsilon$ -greedy 法における探索率
BatchSize	経験再生におけるバッチサイズ
MemorySize	経験再生プールメモリー容量
TargetUpdate	DQN における目標ネットワークの更新頻度
$EPISODES_{Min}^{Complete}$	経験再生状態収集エピソード回数
Evaluation	<i>Evaluation</i> 回数エピソードごとにテスト

---

表6に示した各ハイパーパラメータの役割について、以下に説明する。

*EPISODES* は、エージェントが環境と相互作用しながら学習を行う総エピソード数を表す。この値は、方策の収束性や学習の安定性に影響を与える、エピソード数が不足する場合には十分な学習が行われない一方、過度に大きい場合には計算コストが増大する。エピソード内において、エージェントが環境に対して一度の意思決定を行う単位を *step* と定義する。JSSP における各 *step* では、エージェントは現在の環境を観測し、次に実行する加工工程を選択する、あるいは実行可能な工程が存在しない場合には何も加工を行わない（待機）という行動を選択する。すなわち、1 *step* は、ある時点における加工工程の割り当て判断、または加工を行わない判断に対応する。これらの *step* を繰り返すことで、すべてのジョブの全加工工程が選択完了した時点でエピソードが終了し、当該スケジュールに対する *makespan* が確定する。

$\alpha$  は学習率を表し、ニューラルネットワークのパラメータ更新時における勾配の反映度合いを制御する係数である。学習率の設定は、学習の安定性および収束速度に大きく影響する。

$\gamma$  は将来報酬の重み付け係数であり、現在の行動選択において将来得られる報酬をどの程度考慮するかを決定する。この値が大きいほど長期的な報酬を重視した方策が学習される。本研究の DQN 実装では、メインネットワークにより得られた即時報酬  $r_t$  に対し、ターゲットネットワークを用いて推定した次状態の最大 Q 値  $\max_{a'} Q_{\text{target}}(s_{t+1}, a')$  を  $\gamma$  倍して加算することで、即時的な評価と将来的な評価の両方を考慮した Q 値更新を行っている。

$\varepsilon$  は  $\varepsilon$ -greedy 法における探索率を表す。 $\varepsilon$  は固定値ではなく、エピソードの進行に伴い徐々に減少するよう設定されている。各 *step* において、一様乱数を生成し、その値が  $\varepsilon$  未満の場合には、実行可能な行動集合の中からランダムに行動を選択することで探索を行う。一方、乱数が  $\varepsilon$  以上の場合には、メインネットワークにより推定された Q 値に基づき、実行可能な行動の中で最も Q 値が大きい行動を選択する。このように、学習初期には探索を重視し、学習の進行とともに獲得した知識を活用することで、探索と活用のバランスを調整している。

*BatchSize* は、経験再生において一度の学習更新に用いられる経験データのサンプル数を表す。バッチサイズは、学習の安定性および計算効率に影響を与える。

*MemorySize* は、経験再生プールに保存される状態遷移データの最大容量を表す。この容量が十

---

分でない場合、学習に用いられる経験の多様性が損なわれる可能性がある。

$TargetUpdate$  は、DQN における目標ネットワークの更新頻度を表す。目標ネットワークを一定間隔で更新することで、Q 値推定の不安定化を抑制し、学習の安定性向上を図る。

$EPISODES_{Min}^{Complete}$  は、経験再生に用いる状態遷移データを収集するために必要な、完全なスケジューリングエピソードの最小回数を表す。この設定により、不完全なスケジュールに基づく学習を防止する。

$Evaluation$  は、一定の学習エピソード数ごとに、固定された評価用データセットに対してテストを実施する間隔を表す。これにより、学習過程におけるモデル性能の推移を定期的に確認できる。

#### 6.4.1 複合型報酬関数での訓練

複合型報酬関数を用いた学習では、複数の評価指標を同時に考慮するため、各報酬項に対応する係数の設定が学習挙動および最終的なスケジューリング性能に大きく影響する。本研究では、前節で定義した各報酬項の数式構造は固定した上で、それぞれに付与する係数をハイパーパラメータとして調整し、学習の安定性および得られるスケジュール品質を評価した。

複合型報酬関数に含まれる主な係数と、それぞれが制御する評価項目を以下に示す。

- $w_{time}$ : 加工時間ペナルティの重みを表す係数であり、局所的に加工時間の長い工程選択をどの程度抑制するかを制御する。
- $w_{prog}$ : 加工工程推進報酬の重みを表す係数であり、各 step における工程完了をどの程度強く評価するかを制御する。
- $w_{bal}$ : 機械負荷平準化項の重みを表す係数であり、特定機械への負荷集中をどの程度回避するかを制御する。
- $w_{comp}$ : ジョブ完了時に与えられる完了報酬の大きさを決定する係数である。
- $w_{opp}$ : 機会損失ペナルティの重みを表す係数であり、実行可能な中で短時間で終了する工程を優先させる度合いを制御する。

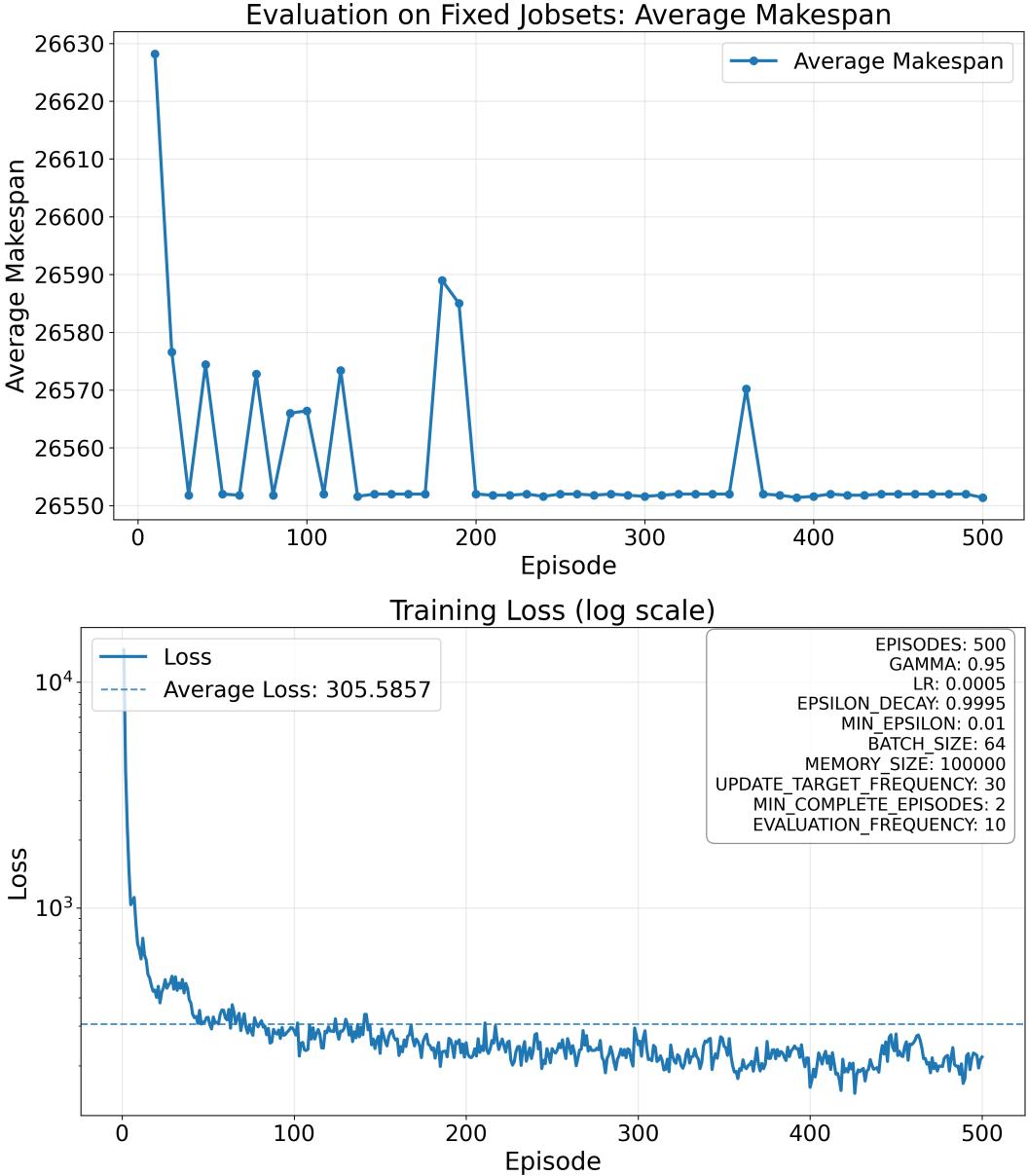


図 10: 複合型報酬関数 DQN での実験

- $w_{final}$ : エピソード終端時に与えられる効率報酬の重みを表す係数であり、スケジュール全体としての *makespan* 短縮をどの程度重視するかを制御する。

図 10 は、複合型報酬関数を用いて学習を行った際の、エピソード数に対する平均 *makespan* および損失関数 (loss) の推移を示したものである。

まず、上段に示す平均 *makespan* の推移に着目すると、学習初期ではエピソードごとのばらつきが大きく、比較的大きな *makespan* が観測されている。これは、 $\epsilon$ -greedy 法によるランダム探索が支配的であり、スケジューリング方策が十分に学習されていないためであると考えられる。そ

の後、学習の進行に伴い平均 *makespan* は段階的に低下し、一定エピソード以降ではほぼ安定した値に収束していることが確認できる。一部のエピソードにおいて一時的な *makespan* の増加が見られるものの、全体としては改善傾向を維持しており、複合型報酬関数に基づく方策学習が有効に機能していることが示唆される。

次に、下段に示す損失関数の推移を見ると、学習初期において loss が急激に減少しており、Q 値推定が速やかに安定方向へ向かっていることが分かる。その後は緩やかな減少傾向を示しながら、一定範囲内で振動する挙動を示している。これは、環境の非定常性および探索行動の影響を受けつつも、学習が発散することなく安定して進行していることを示している。

以上より、複合型報酬関数を用いた場合、平均 *makespan* および損失関数の双方が学習の進行に伴って収束する挙動を示しており、本モデルがスケジューリング方策を段階的に学習していることが確認できる。本研究では、複数のハイパーパラメータ組合せに対して学習を試行し、その中で JSSP に対する性能が最も良好であったモデルの学習過程を図 10 として示している。学習が安定して収束しているにもかかわらず、得られたスケジューリング結果の性能は、図 11 に示すように遺伝的アルゴリズムによる解と比較すると、依然として大きな差が存在することが確認された。

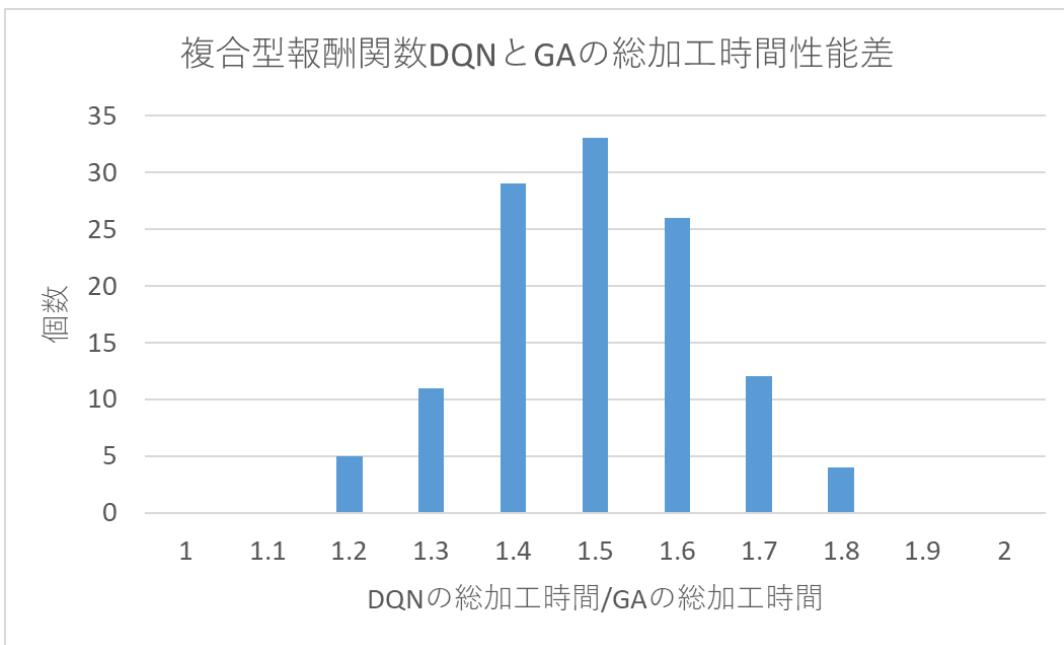


図 11: 複合型報酬関数 DQN と GA の総加工時間性能差

このことで、複合型報酬関数を用いた場合、報酬に影響を与える要因が多岐にわたるため、各

---

step における報酬信号が複雑かつノイズを含みやすくなることが分かる。その結果、学習過程において損失関数や平均 makespan の収束が確認される一方で、エージェントがスケジューリングにおける本質的な改善方策を十分に獲得できているとは限らず、見かけ上は学習が進行しているよう見えても、実際の問題解決能力には必ずしも結び付かない場合がある。

そこで次節では、報酬構造を簡略化した *makespan* 中心型報酬関数について検討する。

#### 6.4.2 *makespan* 中心型報酬関数での訓練

本小節では、複合型報酬関数とは異なり、最終的なスケジューリング性能指標である *makespan* の改善に焦点を当てた、*makespan* 中心型報酬関数を用いた学習について述べる。

本报酬設計では、複数の評価項目を同時に考慮する複合型報酬関数と比べ、報酬信号を可能な限り単純化し、各 step における意思決定が最終的な *makespan* に与える影響を直接的に学習させることを目的としている。具体的には、step 前後における総加工時間  $C_{\max}$  の変化量に基づき、 $C_{\max}$  の増加を抑制する行動に対して正の評価を与え、逆に  $C_{\max}$  を増大させる行動には負の評価を与える構成とした。

*makespan* 中心型報酬関数では、最終的な総加工時間の抑制を主目的としつつ、学習の停滞や非効率な行動を防ぐため、いくつかの補助的な報酬項およびペナルティ項を導入している。これらの各項に対応する係数を以下に示す。

- $w_{\Delta C}$ : *makespan* の増分  $\Delta C_{\max}$  に対する重みを表す係数であり、各 step において総加工時間の増加をどの程度強く抑制するかを制御する。本报酬関数における主成分である。
- $w_{\text{idle}}$ : 機械の待機時間に対するペナルティの重みを表す係数であり、不必要的待機や加工開始の遅延を抑制する役割を持つ。
- $w_{\text{miss}}$ : 実行可能な加工工程が存在するにもかかわらず、それを選択しなかった場合に与えられるペナルティの重みを表す係数であり、機会損失を明示的に評価する。
- $w_{\text{skip}}$ : 待機行動 ( $\text{action} = 0$ ) に対する軽微なペナルティの重みを表す係数であり、意思決定の回避を過度に選択することを防ぐ目的で導入する。

- 
- $w_{imm}$ : 最も早く空いた機械において、即時に加工を開始できた場合に与えられる補助的報酬の重みを表す係数であり、機械資源の有効活用を促進する。

図 12 は、*makespan* 中心型報酬関数を用いて学習を行った際の、エピソード数に対する平均 *makespan*、損失関数 (*loss*) と平均総報酬の推移を示したものである。

図 12 から分かるように、複合型報酬関数の場合とは異なり、いずれの観測指標においても明確な収束傾向は確認されない。

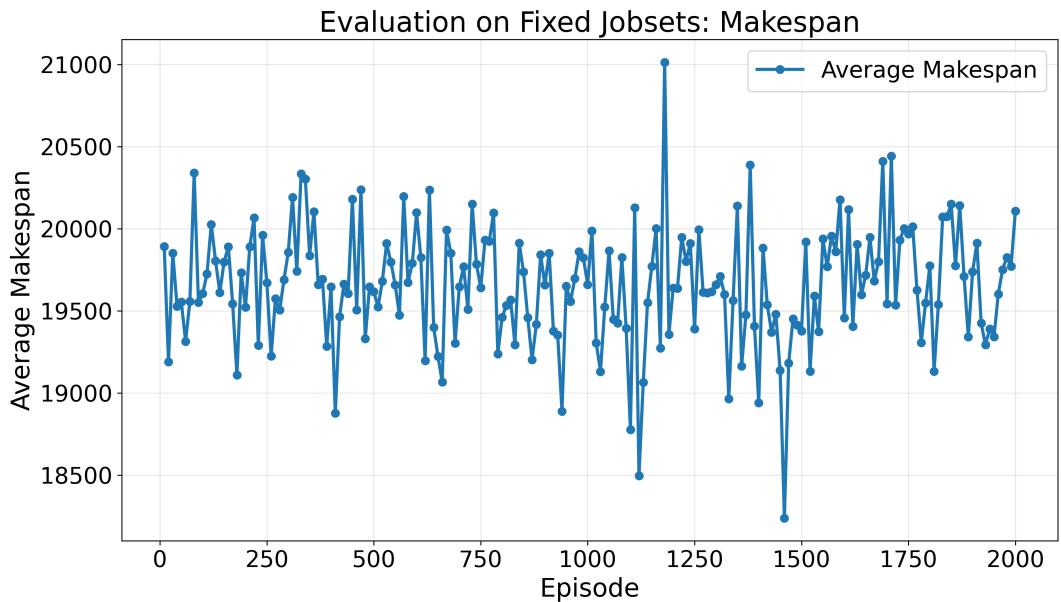
特に、平均 *makespan* および平均総報酬は、エピソードの進行に伴って大きなばらつきを維持しており、学習が単調に改善方向へ進行しているとは言はずらしい。また、損失関数についても、学習全体を通じて大きな変動が観測されており、Q 値推定が安定的に収束しているとは言えない動きを示す。

ですが、本モデルを用いて実際の JSSP に対するスケジューリングを行った結果、図 13 に示すように、得られた解の性能は、GA による解と互角している、さらに一部の問題例においては、GA によって得られた解を上回る場合も確認された。このことは、学習過程における評価指標の収束性と、最終的な問題解決能力とが必ずしも一致しないことがわかる。

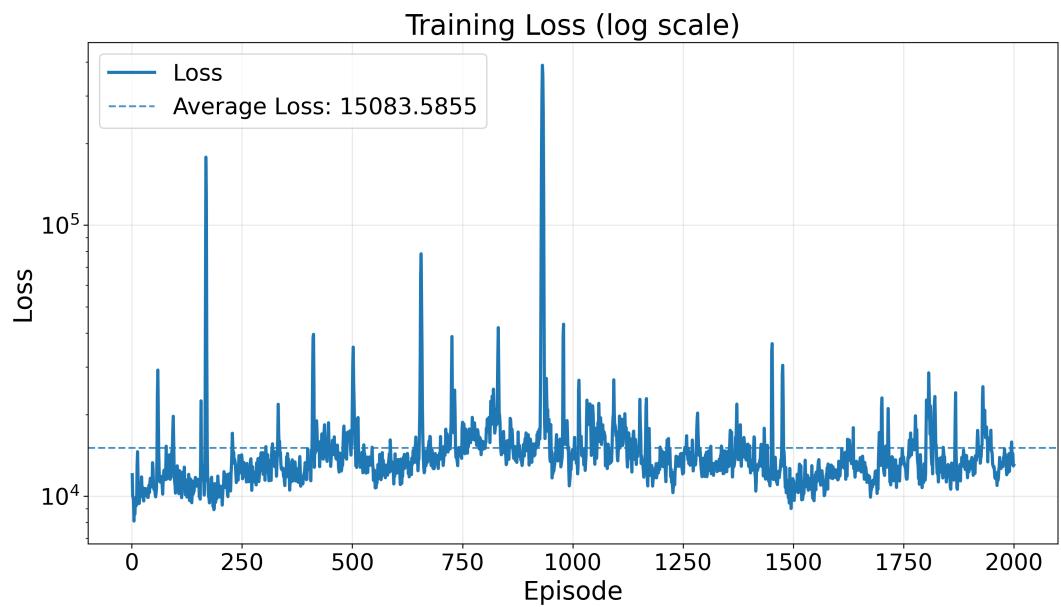
*makespan* 中心型報酬関数では、報酬関数が  $C_{\max}$  の増分に直接基づいて与えられるため、局所的な意思決定の良否が即時的に評価される一方、環境の非定常性により、学習過程全体としては不安定な挙動を示しやすい。しかしながら、このような報酬設計は、スケジューリングにおける有効なヒューリスティックをネットワーク内部に獲得させるには十分であり、結果として実用上有効な解を導出できたものと考えられる。

このような、学習過程において損失関数や評価指標が明確に収束しない一方で、最終的に得られる解の性能が一定水準に達するという挙動は、強化学習において必ずしも例外的な現象ではない。Sutton および Barto は、強化学習における価値関数学習では、損失関数 (TD 誤差など) が教師あり学習における誤差指標とは本質的に異なり、学習の進行とともに環境分布や方策分布が変化するため、単調な収束を示さない場合が多いことを指摘している [24]。そのため、損失関数の挙動のみから、最終的な方策性能を直接評価することは困難である。

さらに、Tsitsiklis および Van Roy は、関数近似とブートストラップを併用する価値関数学習に

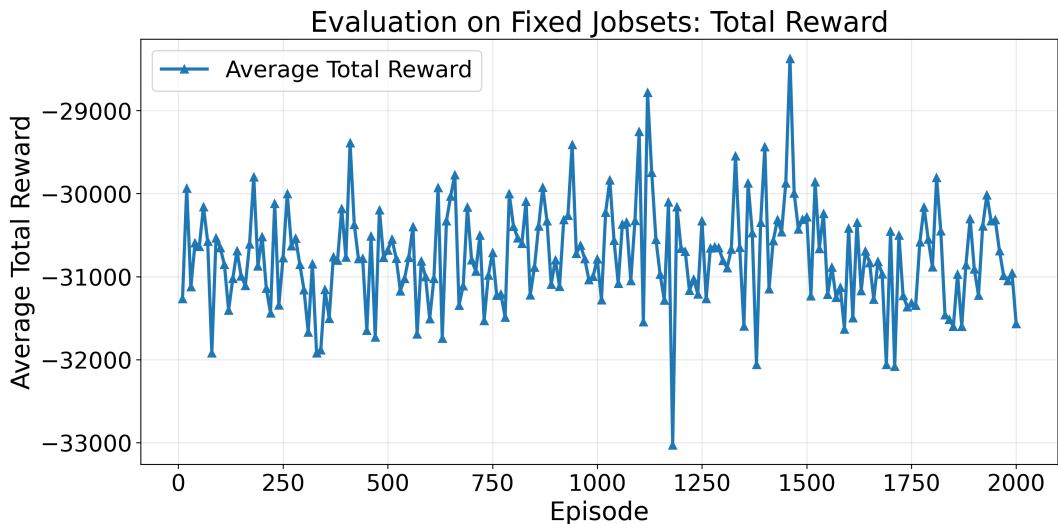


(a) 中心型報酬関数 DQN での実験の *makespan* 評価



(b) 中心型報酬関数 DQN での実験の loss 関数

図 12: 中心型報酬関数 DQN での実験



Hyperparameters

```

EPISODES: 2000
GAMMA: 0.9
LR: 0.0005
EPSILON_DECAY: 0.998
MIN_EPSILON: 0.01
BATCH_SIZE: 64
MEMORY_SIZE: 100000
UPDATE_TARGET_FREQUENCY: 30
MIN_COMPLETE_EPISODES: 2
EVALUATION_FREQUENCY: 10
    
```

(c) 中心型報酬関数 DQN での実験の平均総報酬

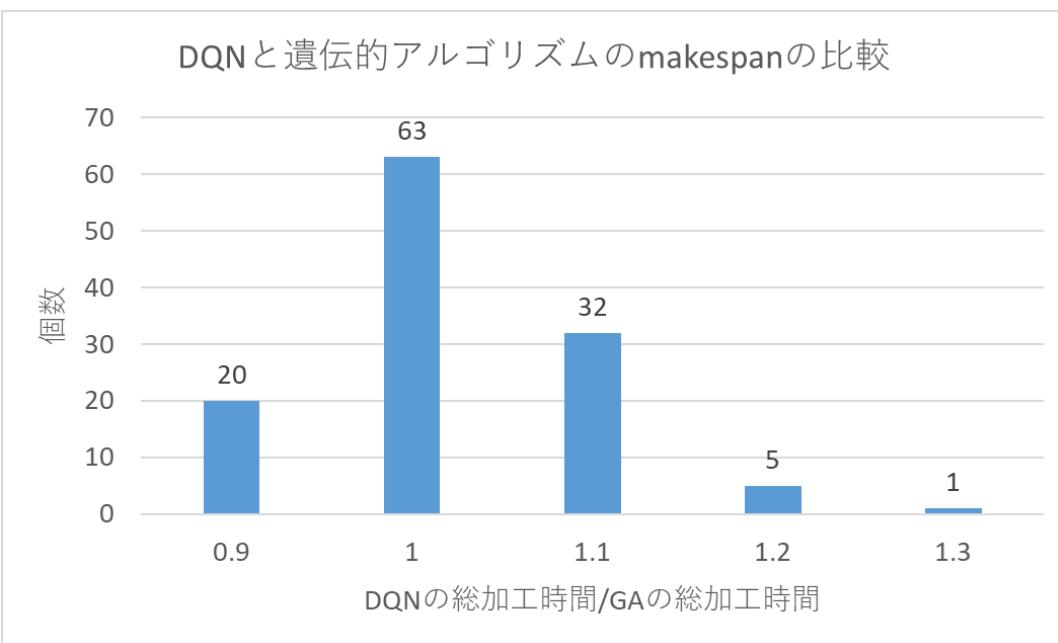


図 13: 中心型報酬関数 DQN と GA の総加工時間性能差

---

においては、理論的にも学習過程が振動または非収束となる可能性があることを示している [28]。この結果は、DQN のようなオフポリシー学習に基づく手法において、損失関数の安定性が保証されない状況下でも、必ずしも方策が無意味になるわけではないことを示唆している。

また、JSSP を対象とした深層強化学習の研究においても、学習過程の収束性よりも、最終的に得られるスケジューリング解の品質が重視される傾向が報告されている [26]。Zhang らは、学習曲線が大きく変動する場合であっても、適切なディスパッチング方策が獲得されれば、従来のヒューリスティック手法と同等の性能が得られることを示している。

以上の文献を踏まえると、本研究において観測された「学習指標の非収束性」と「実問題に対する解探索能力」との乖離は、*makespan* 中心型報酬関数の特性と、強化学習における価値関數学習の性質に起因するものと考えられる。すなわち、本報酬設計は、安定した学習曲線の獲得には課題を残すものの、スケジューリングに有効なヒューリスティックをネットワーク内部に形成するには十分であり、その結果として GA と同程度、あるいは一部問題においてそれを上回る解が得られたと解釈できる。

## 6.5 訓練済みモデルでの実験結果

本章で学習したモデルを用いて得られたスケジューリング解と、GA によって得られた解とを比較し、*makespan* およびスケジューリング構造の観点から、両手法の性能差について詳細に検討する。

前章に述べた 3 タイプの注文(小、並、大)ごとにに対して、複合型報酬関数での DQN と中心型報酬関数での DQN 両方を GA と比較する。複合型報酬関数での DQN で得られた解はほとんど GA に超えられないで、3 タイプの注文ごとにそれぞれ GA と比べて近い例(比率 1.3 ぐらい)、離れた例(比率 1.5 ぐらい)とだいぶ離れた例(比率 1.8 ぐらい)を示す。中心型報酬関数での DQN で得られた解は GA と互角しているので、3 タイプの注文ごとにそれ GA を超える例(比率 0.9 ぐらい)、GA 相当な例(比率 1 ぐらい)と GA を劣る例(比率 1.2 ぐらい)を示す。

### 6.5.1 複合型報酬関数での実験結果

訓練済みモデルで実験用データである 120 個ジョブセットに対して, 得られた *makespan* は図 14 に示す.

<b>id</b>	<b>タイプ</b>	<b>複合型DQN makespan</b>						
1	regular	28195	41	large	30495	81	small	22485
2	regular	29590	42	small	21090	82	large	27790
3	regular	19715	43	small	23885	83	small	17665
4	regular	27330	44	regular	26240	84	regular	24565
5	small	19810	45	small	14170	85	regular	26320
6	regular	22475	46	large	32230	86	large	29025
7	regular	29425	47	regular	31850	87	large	29210
8	regular	30005	48	large	28975	88	large	30960
9	small	23985	49	regular	28765	89	small	18625
10	small	26350	50	small	24090	90	large	29425
11	small	21960	51	regular	24405	91	regular	22385
12	large	34400	52	large	29250	92	regular	21705
13	small	17710	53	regular	20665	93	large	35245
14	small	19175	54	small	23855	94	small	20925
15	small	19290	55	small	25000	95	large	32340
16	regular	28125	56	small	21315	96	regular	29875
17	regular	29895	57	large	32925	97	small	21810
18	regular	23630	58	large	32215	98	small	24710
19	small	23960	59	regular	24055	99	small	19790
20	large	30335	60	large	31370	100	large	25255
21	regular	23130	61	large	27615	101	regular	24745
22	regular	23795	62	regular	26450	102	regular	33265
23	large	27165	63	large	29605	103	large	38105
24	large	28495	64	small	21945	104	small	24035
25	large	34485	65	small	23145	105	large	31145
26	small	17145	66	regular	27320	106	large	30940
27	large	26370	67	large	37675	107	small	17820
28	regular	25535	68	regular	34290	108	large	30070
29	small	23025	69	small	15940	109	small	18810
30	large	32400	70	regular	27885	110	small	27465
31	large	38910	71	large	35365	111	large	29565
32	small	22730	72	regular	28940	112	small	19365
33	regular	23860	73	large	35395	113	small	19510
34	regular	28455	74	large	33495	114	small	19760
35	small	19290	75	small	22440	115	regular	29700
36	large	27755	76	large	31935	116	small	19580
37	regular	22295	77	large	32970	117	large	29530
38	regular	27180	78	regular	28610	118	large	30435
39	regular	24410	79	small	21705	119	large	30255
40	regular	24795	80	small	21410	120	regular	25105

図 14: 複合型報酬関数で訓練済みモデルを用いて実験用データの実験結果

以下の図 15 から図 33 までの図は, 実験用データにあるジョブセットに対して, GA で得られた最適解ガントチャートと DQN(複合型報酬関数 DQN と *makespan* 中心型報酬関数) で得られた最適解ガントチャート及び GA と DQN それぞれが結果出るかかる時間のデータが記載されている. 縦軸は機械番号, 横軸は時刻をとっている.

注文のタイプごとに複合型報酬関数 DQN の性能について個例を挙げると.

図15は注文(小)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.34の例。GAの例を観察すると、M4とM5の使い方で、QAとDQNで大きな違いがあり、GAは機械のアイドル時間なく使っているが、DQNはそれぞれ時間5000 7500、2200 8500に大きな空白があり、ジョブ6の加工工程が全部完成するまで、他のジョブ加工プロセスを選択せず。その間の作業がないために、後続するの加工プロセスは以降に大きな遅れがでてしまっていることがわかる、そのためmakespanがDQNのほうが長くかかってしまった例である。

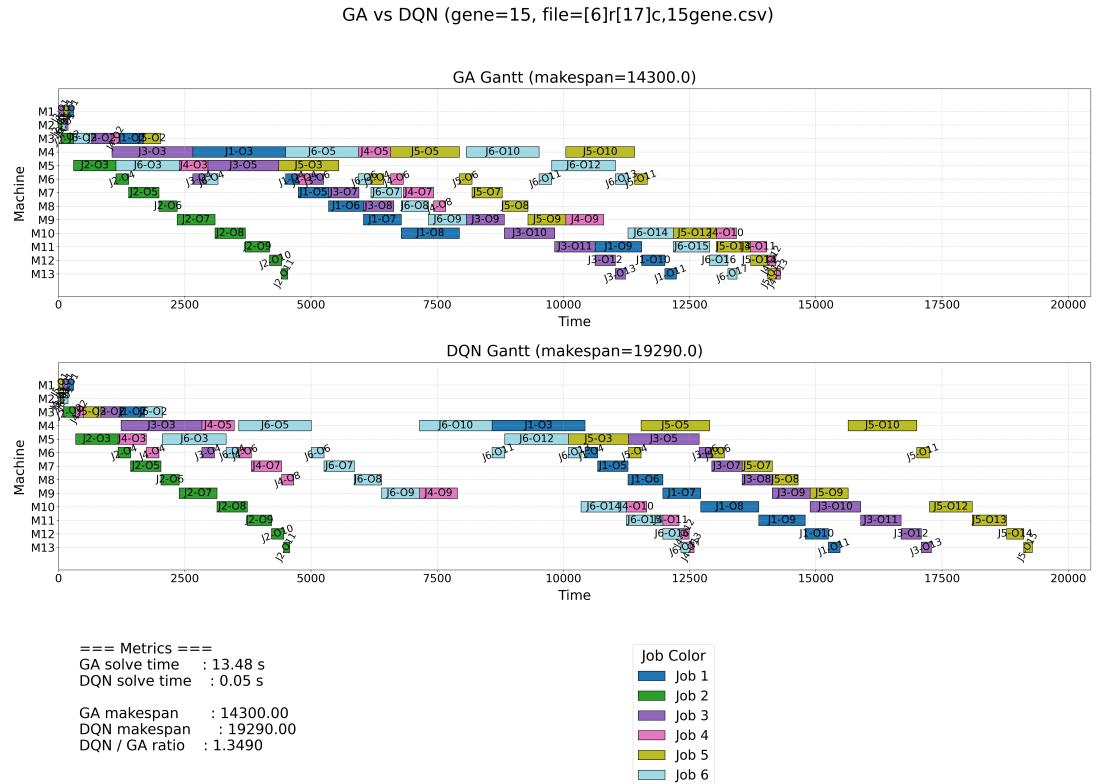


図15: 注文(小)複合型報酬関数DQNとGAのガントチャート(比率1.34)

図16は注文(小)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.48の例。GAの例を観察すると、またジョブ6とジョブ5の取り扱うがQAとDQNで大きな違いがあり、ジョブ6の加工工程が全部完成するまで、他のジョブ加工プロセスを選択せず、ジョブ5の加工工程が全部スケジュール案の最後に配置され、機械のアイドル時間ほどんど使用しないことでmakespanがDQNのほうが長くかかってしまった例である。

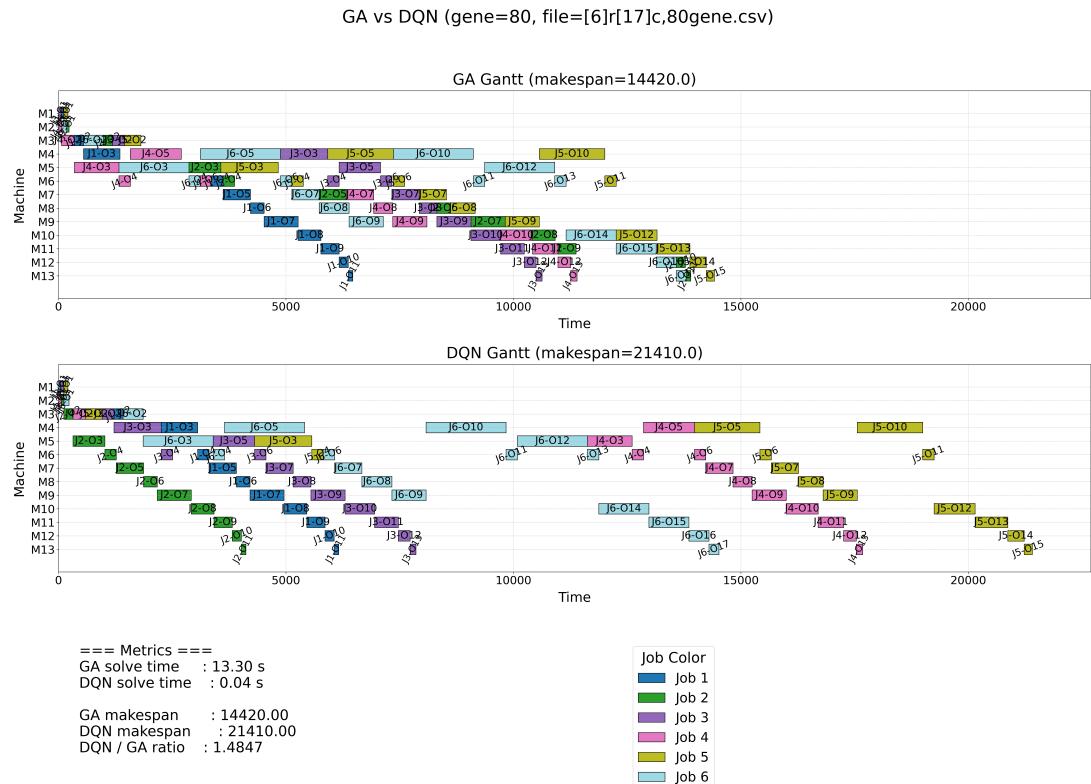


図 16: 注文(小)複合型報酬関数DQNとGAのガントチャート(比率1.48)

図17は注文(小)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.76の例。今回の例は、DQNのスケジュール案はジョブ5とジョブ6共にスケジュールが完成するまで、他のジョブの加工プロセスを一切配置しないことで、時間を無駄、DQNのmakespanはGAのmakespanの二倍弱にした。

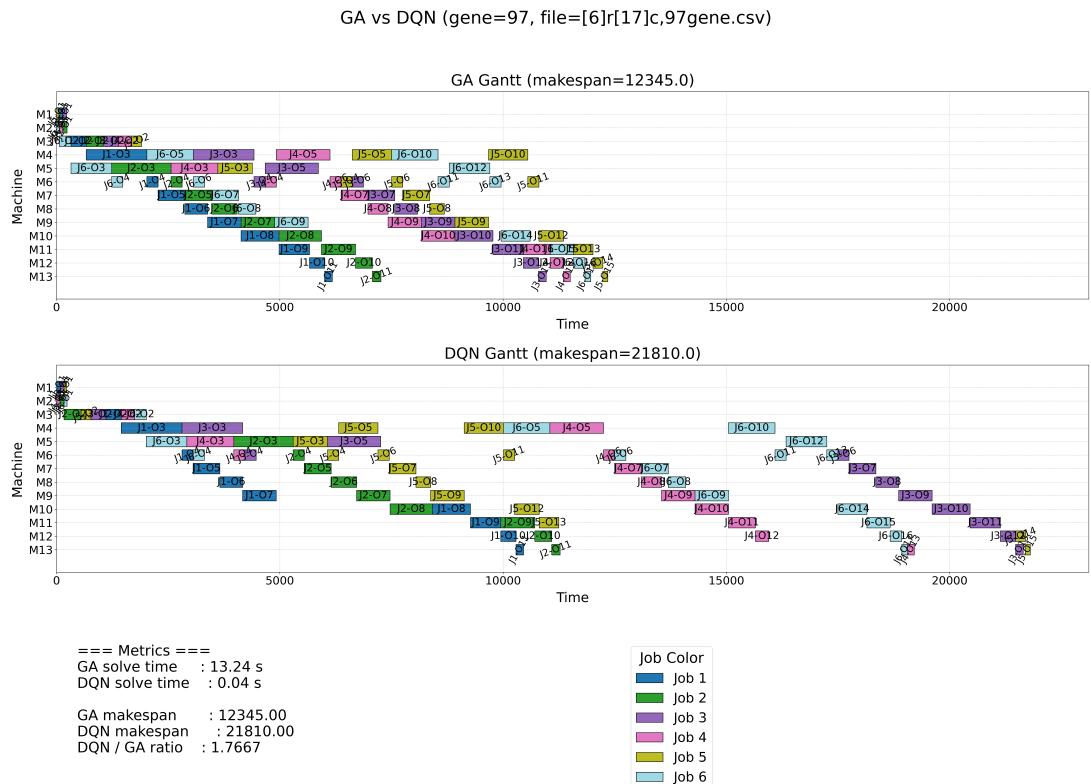


図17: 注文(小)複合型報酬関数DQNとGAのガントチャート(比率1.76)

図18は注文(並)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.2の例。この例でしたら、DQNがジョブ6とジョブ5の加工工程に他のジョブ加工プロセスを配置することで、GAとの差を小さくした。だが、まだ改善できるところがある。例えばM10以後の機械の使い方、GAはM10以後の機械をまとめて加工する。DQNは各加工工程の間に遊び時間を入れている。

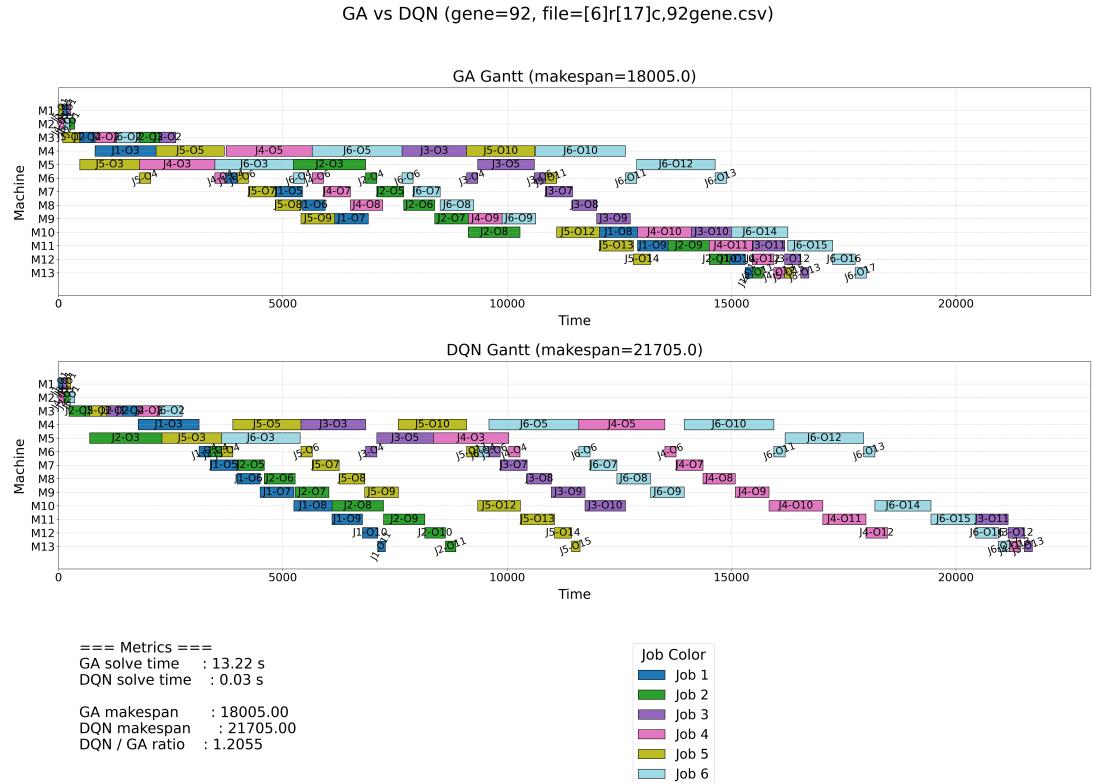


図18: 注文(並)複合型報酬関数DQNとGAのガントチャート(比率1.2)

図19は注文(並)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.5の例。この例のスケジュールパターンは図17で示した例と同じ、ジョブ5とジョブ6の加工工程配置が不適切である。

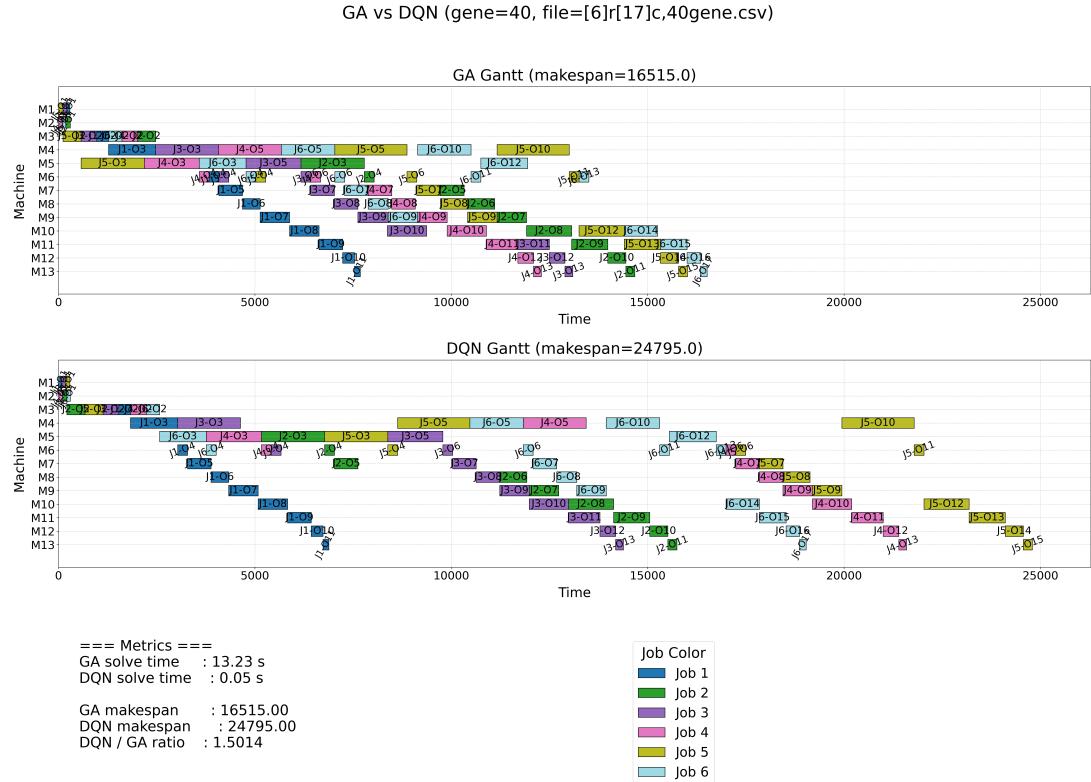


図19: 注文(並)複合型報酬関数DQNとGAのガントチャート(比率1.5)

図 20 は注文 (並) タイプに対して、複合型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 1.72 の例。この例もは図 17 で示した例と同じように設置されて、*makespan* を長伸ばした。

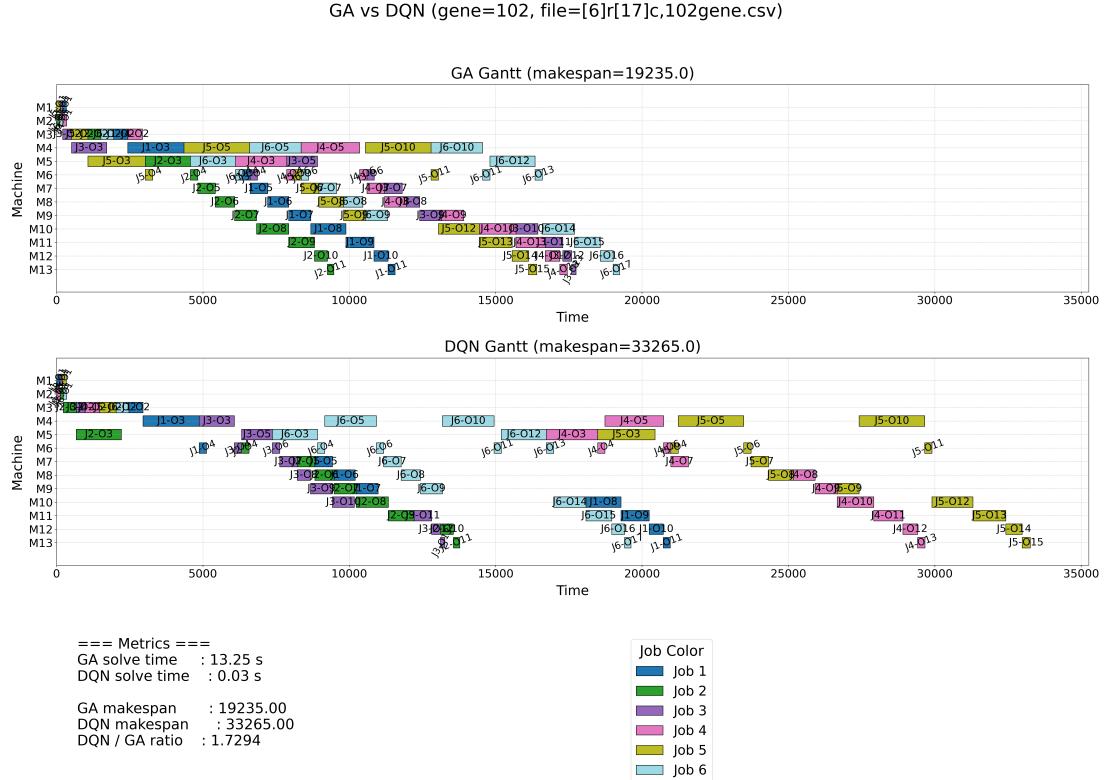


図 20: 注文 (並) 複合型報酬関数 DQN と GA のガントチャート (比率 1.72)

図21は注文(大)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.22の例。この例でしたら、ジョブ5の中にジョブ6を入れていたが、ジョブ6の加工工程が全部完成するまで他のジョブの加工プロセスを入れなかった。

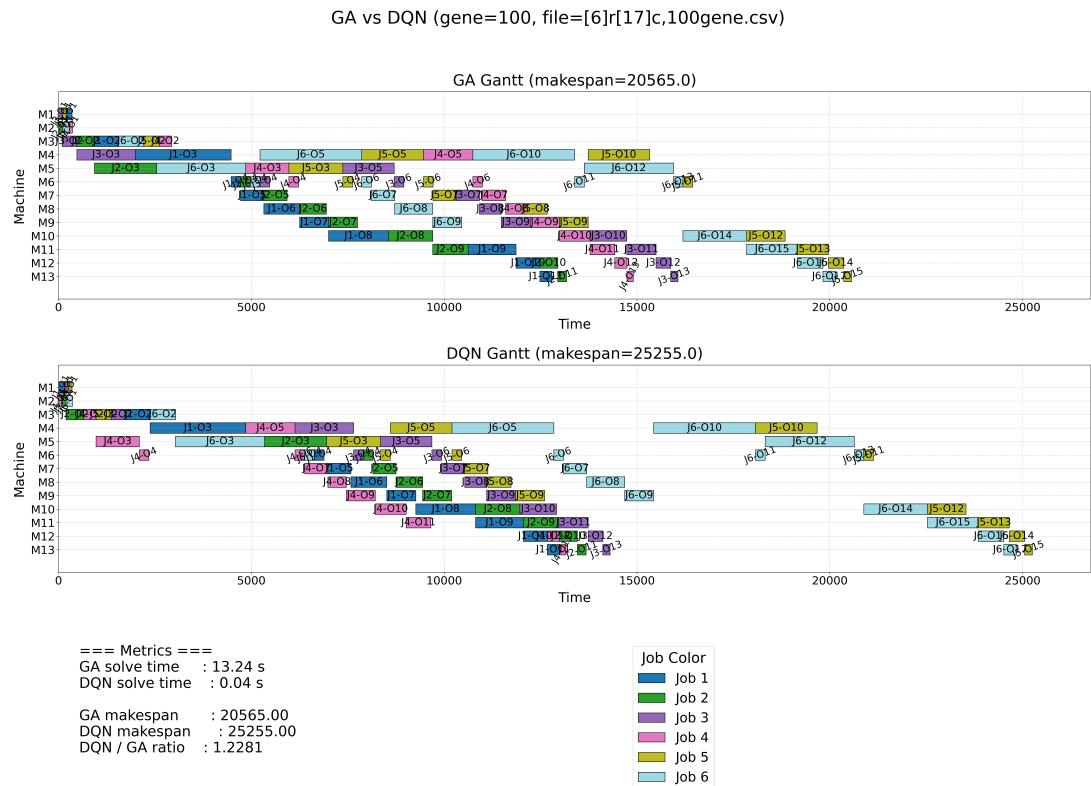


図 21: 注文(大)複合型報酬関数 DQN と GA のガントチャート(比率 1.22)

図22は注文(大)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.5の例。この例も先の図20同じのパターンでよくスケジュールできなかった。

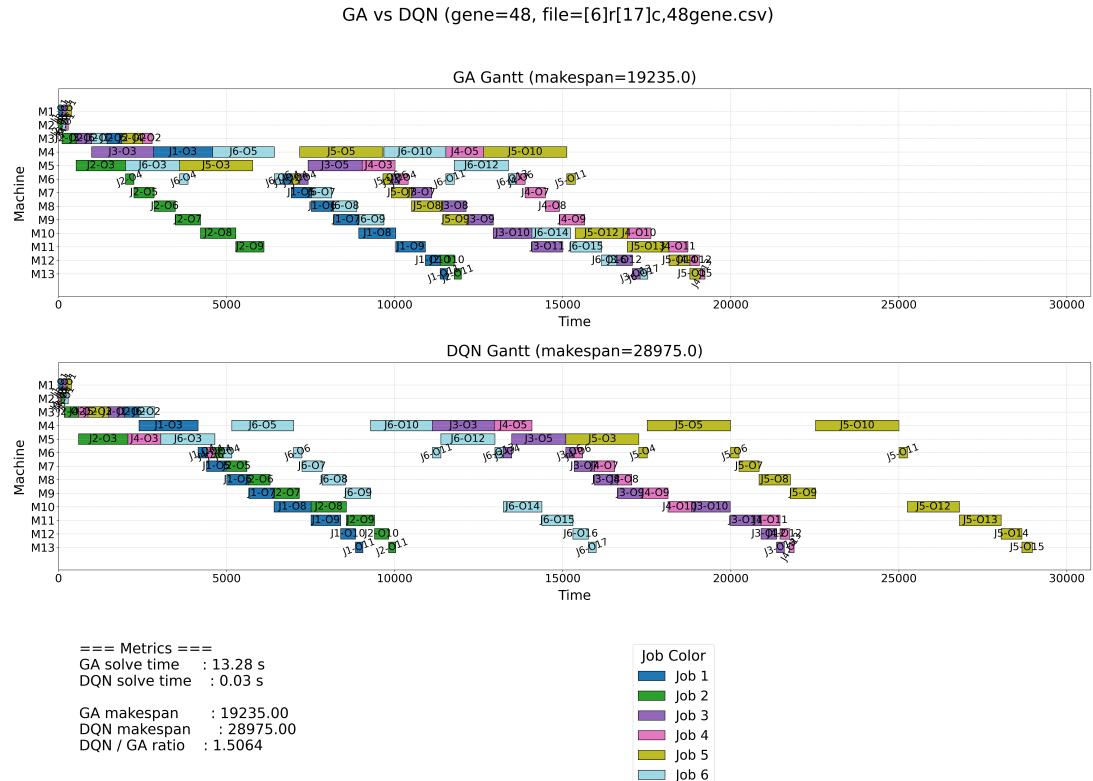


図22: 注文(大)複合型報酬関数DQNとGAのガントチャート(比率1.5)

図23は注文(大)タイプに対して、複合型報酬関数DQNで得られたガントチャートのmakespanとGAで得られたガントチャートのmakespanの比率が1.71の例。最後の複合型報酬関数DQNとGAのガントチャートの例も、ジョブ5とジョブ6を除いて、隙間なく仕事しているが、ジョブ6とジョブ5で時間を無駄遣いした。

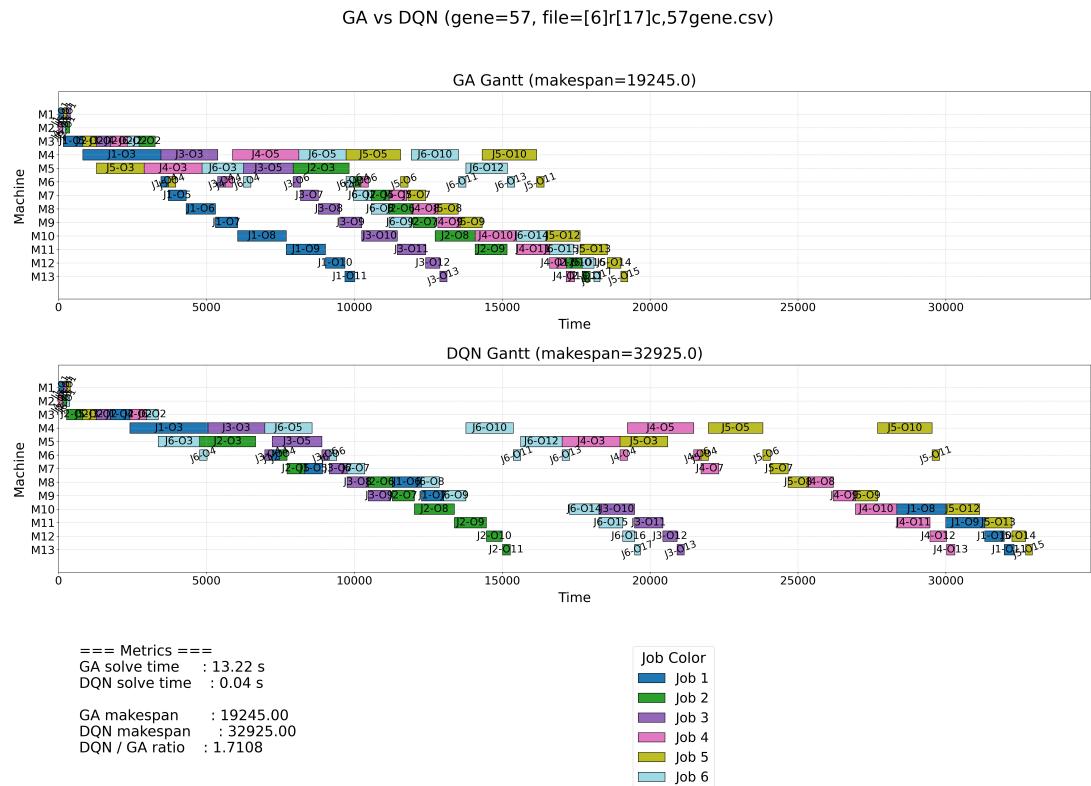


図 23: 注文(大)複合型報酬関数DQNとGAのガントチャート(比率1.71)

---

図群に示す複合型報酬関数を用いた DQN のスケジューリング結果では, GA により得られた解と比較して *makespan* が大きく悪化することが確認された. 特に, 工序数が多い Job5 および Job6 に処理が集中し, 他ジョブの実行可能な工序が十分に挿入されないスケジュール構造が共通して観測される.

この要因の一つとして, 本研究で設計した複合型報酬関数における即時報酬と将来報酬の相対的な影響の偏りが挙げられる. 本環境では, 残作業量が最大のジョブを優先する項が強く作用しており, Job5 および Job6 を連続的に選択する行動系列は, 将来の報酬を含めた行動価値として高く評価されやすい. その結果, ある時点で Job5 または Job6 を選択した後は, 同一ジョブを継続して処理する行動が高い将来報酬を見込める選択として学習される.

一方で, 空き機械を利用して他ジョブの工序を挿入する行動は, その時点で得られる即時報酬が比較的小さい. たとえ長期的には待ち時間の削減や *makespan* の短縮に寄与する可能性があったとしても, 当該行動に対する即時報酬が, 直前まで選択されていた Job5 または Job6 の継続処理によって期待される将来報酬を上回らない場合,DQN はこれを選択しにくい. すなわち, 「他ジョブを挿入する」という行動は, 即時報酬の観点から不利であるだけでなく, 既に形成された行動系列に基づく将来報酬と比較しても行動価値が低く評価される傾向にある.

このように, 即時報酬が小さいが長期的には有効な行動と, 即時および将来報酬の両面で高く評価されやすい行動との間で報酬構造に非対称性が存在する場合, エージェントは後者に過度に依存した方策を学習しやすい. その結果, Job5 および Job6 の処理が過度に優先され, 他ジョブの工序を柔軟に挿入する探索が十分に行われず, GA が実現している空き時間活用型のスケジューリングと比較して *makespan* が悪化したものと考えられる.

### 6.5.2 *makespan* 中心型報酬関数での実験結果

訓練済みモデルで実験用データである 120 個ジョブセットに対して, 得られた *makespan* は図 24 に示す.

注文のタイプごとに *makespan* 中心型報酬関数 DQN の性能について個例を挙げると.

図 25 は注文 (小) タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 0.95 の例. *makespan* 中心型報

id	タイプ	単純型DQN makespan						
1	regular	19050	41	large	21220	81	small	15085
2	regular	20935	42	small	12525	82	large	20285
3	regular	18025	43	small	14120	83	small	10515
4	regular	21390	44	regular	19615	84	regular	19010
5	small	12420	45	small	9130	85	regular	19950
6	regular	16970	46	large	26455	86	large	20800
7	regular	19495	47	regular	18445	87	large	23555
8	regular	21295	48	large	19090	88	large	19920
9	small	16545	49	regular	19430	89	small	13230
10	small	16880	50	small	15025	90	large	21345
11	small	14335	51	regular	16320	91	regular	18650
12	large	28500	52	large	18710	92	regular	22055
13	small	13155	53	regular	16095	93	large	23470
14	small	14165	54	small	14770	94	small	15270
15	small	15070	55	small	13935	95	large	23545
16	regular	17240	56	small	16435	96	regular	18525
17	regular	17030	57	large	19675	97	small	12015
18	regular	19210	58	large	21695	98	small	12640
19	small	16180	59	regular	17430	99	small	16700
20	large	23620	60	large	20795	100	large	23060
21	regular	16930	61	large	22485	101	regular	17640
22	regular	15100	62	regular	19735	102	regular	21385
23	large	22985	63	large	21810	103	large	25265
24	large	20315	64	small	16330	104	small	13890
25	large	21525	65	small	14725	105	large	20590
26	small	10695	66	regular	18505	106	large	22000
27	large	21210	67	large	23990	107	small	10985
28	regular	20135	68	regular	20630	108	large	22975
29	small	13355	69	small	11970	109	small	12250
30	large	24040	70	regular	18475	110	small	15320
31	large	25085	71	large	21690	111	large	18975
32	small	13790	72	regular	21120	112	small	14710
33	regular	21870	73	large	27605	113	small	13045
34	regular	21045	74	large	19410	114	small	14950
35	small	11735	75	small	13190	115	regular	20065
36	large	23885	76	large	21220	116	small	13370
37	regular	16970	77	large	21675	117	large	19380
38	regular	18245	78	regular	18900	118	large	20655
39	regular	16230	79	small	14920	119	large	19965
40	regular	17215	80	small	14830	120	regular	21085

図 24: 複合型報酬関数で訓練済みモデルを用いて実験用データの実験結果

酬関数 DQN 最初の例から、複合型報酬関数 DQN で得られたガントチャートのスケジュール案と比べて、隙間なく加工プロセスを配置できました。複合型報酬関数 DQN が苦手なジョブ 5 とジョブ 6 についても、上手く他の加工プロセスを入れることで、GA により短い *makespan* のスケジュール案を得られました。

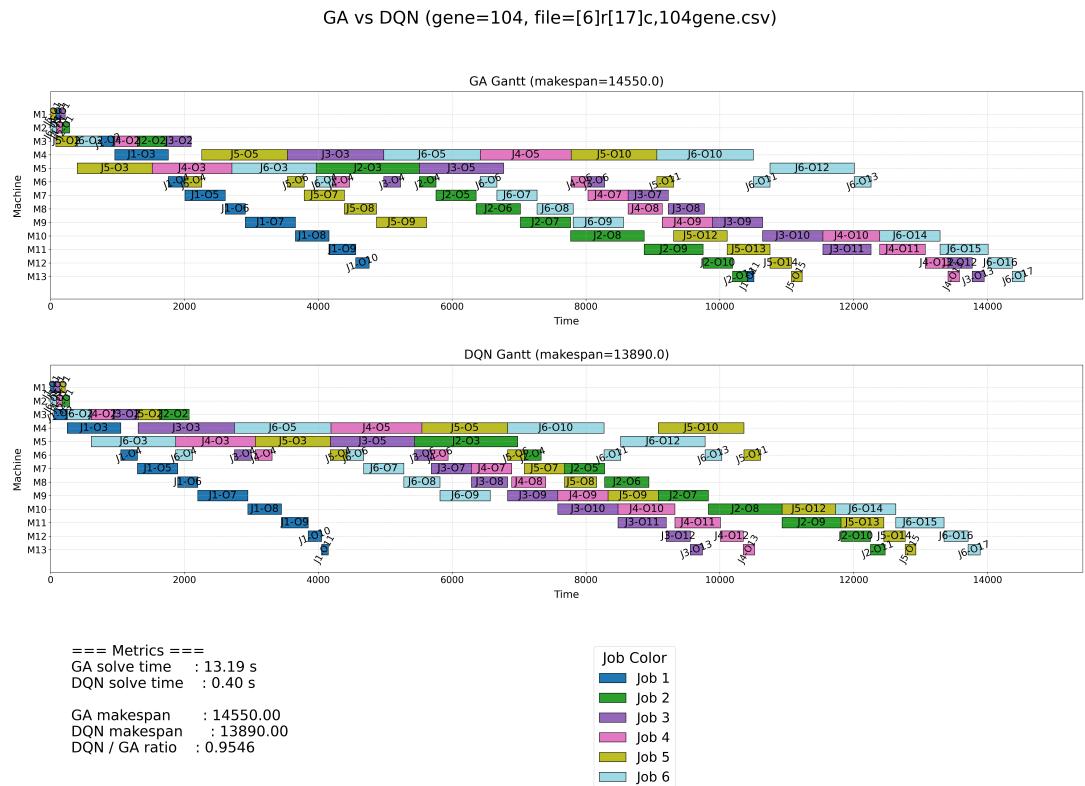


図 25: 注文 (小)*makespan* 中心型報酬関数 DQN と GA のガントチャート (比率 0.95)

図 26 は注文 (小) タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 0.99 の例. この例も *makespan* 中心型報酬関数 DQN がほんの少し優れた例である. 全対の加工プロセスが配置された形とすると, GA と DQN がほぼ同じ方策で配置している.

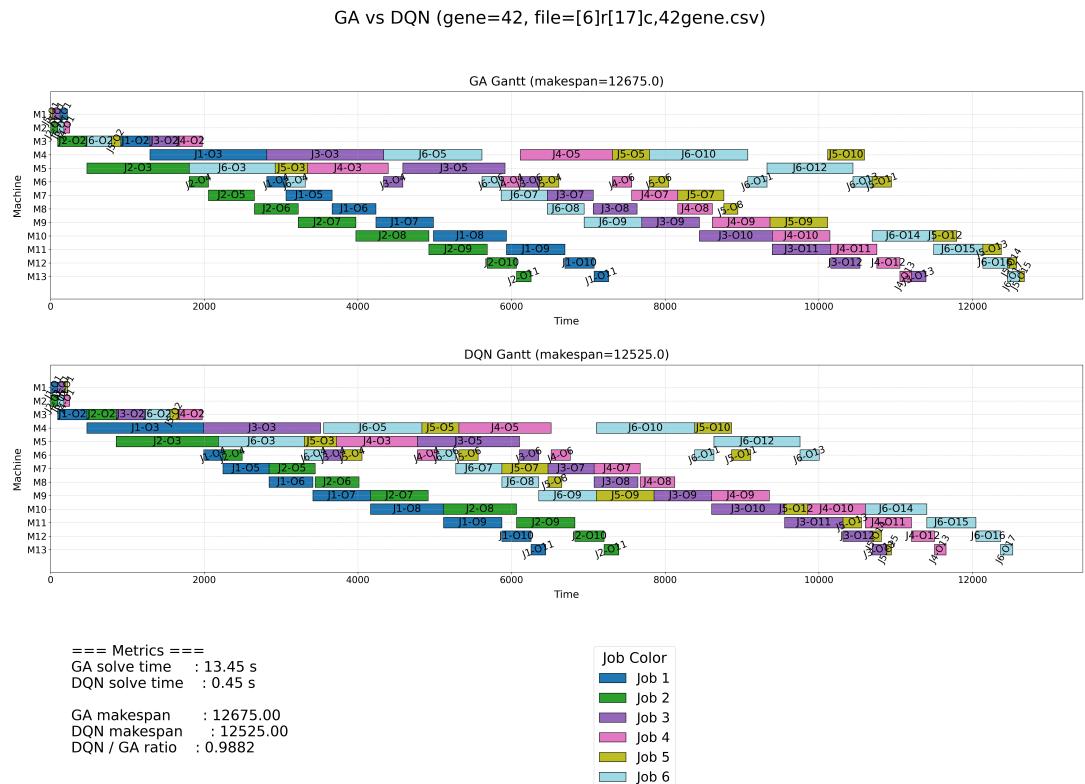


図 26: 注文 (小)*makespan* 中心型報酬関数 DQN と GA のガントチャート (比率 0.99)

図27は注文(小)タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 1.24 の例. 今回挙げられた例は, DQN が GA と比べて, かなり劣るになった. ジョブ 5 とジョブ 6 を除いてジョブを詰めて加工し, ジョブ 5 の加工プロセスの間に少しジョブ 6 の加工プロセスを入れて, ジョブ 6 を最後の集中加工した. これにより, *makespan* を伸ばした.

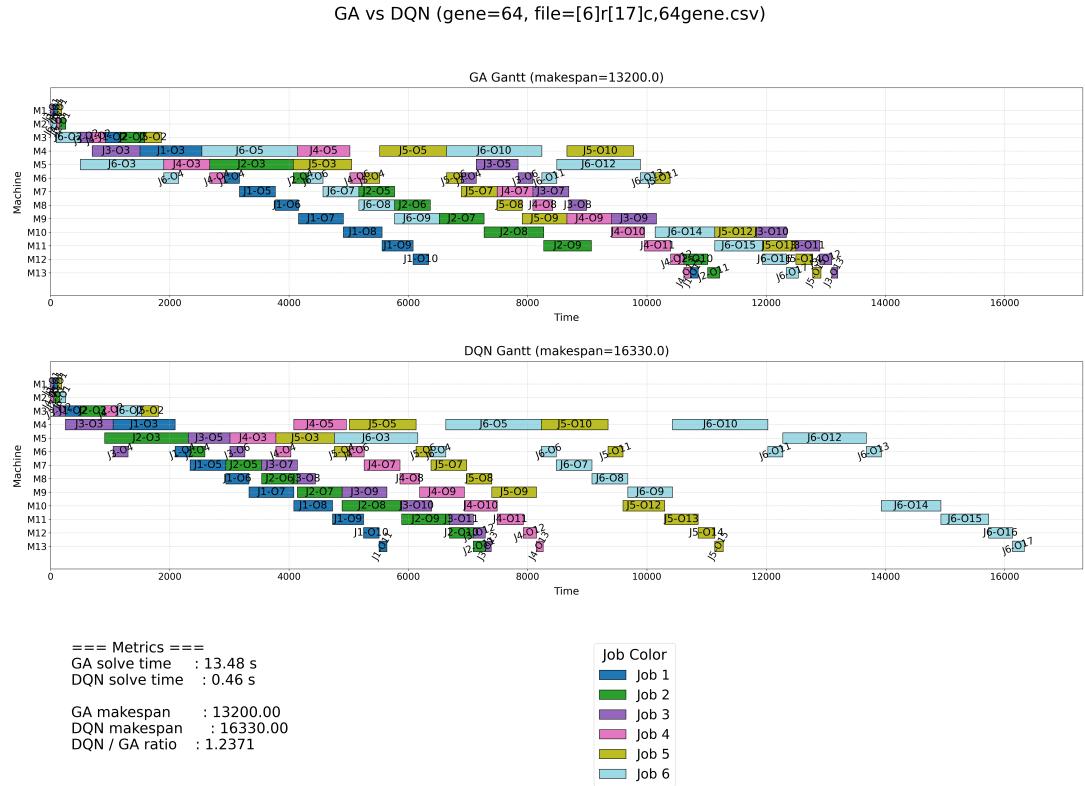


図 27: 注文(小)*makespan* 中心型報酬関数 DQN と GA のガントチャート(比率 1.24)

図 28 は注文 (並) タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 0.97 の例. この DQN スケジュール案は, 6 種類のジョブに一番短いジョブ 1 を真っ先に加工させ, GA と比べて *makespan* を短縮した.

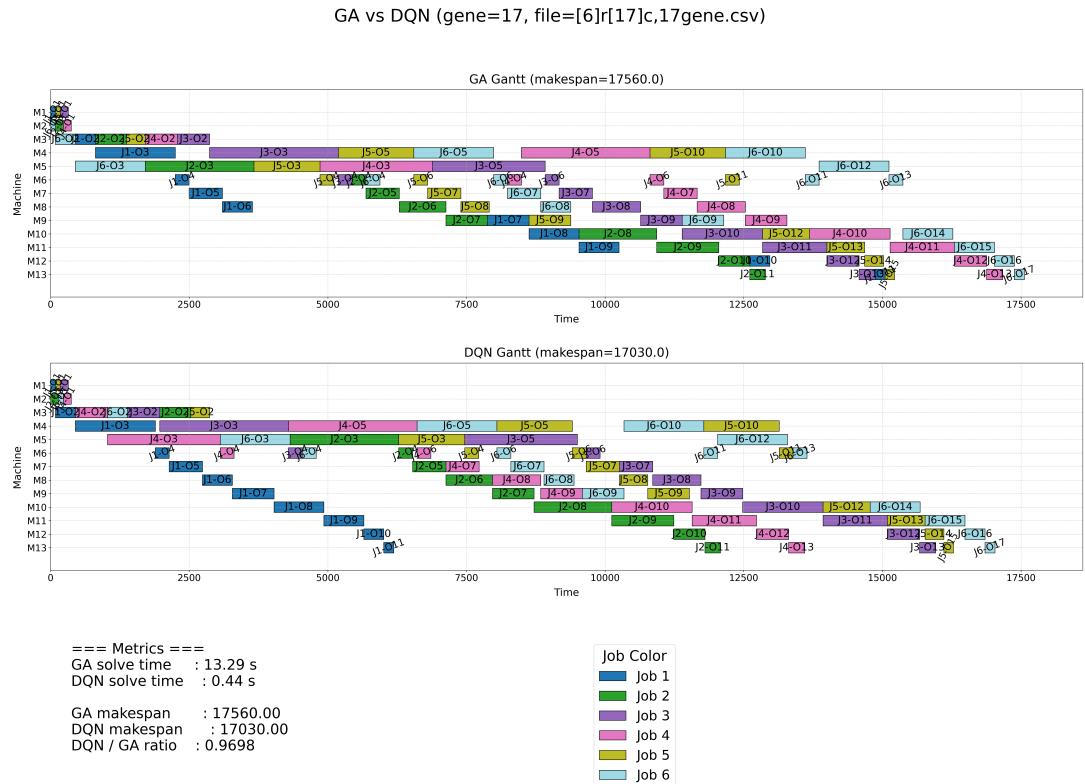


図 28: 注文 (並) *makespan* 中心型報酬関数 DQN と GA のガントチャート (比率 0.97)

図29は注文(並)タイプに対して, makespan 中心型報酬関数 DQN で得られたガントチャートの makespan と GA で得られたガントチャートの makespan の比率が 0.99 の例. このジョブセットに関する GA スケジュール案と DQN スケジュール案は makespan の長さはほぼ同じですが, GA のスケジュール案はプロセスの間に十分のアイドル時間を入れ, 逆に DQN の方はプロセスを詰めて集中加工傾向がある. これにより, GA で得られたスケジュール案の特徴は DQN と比べて, 前に挙げられた例と真っ逆のパターンも確認した.

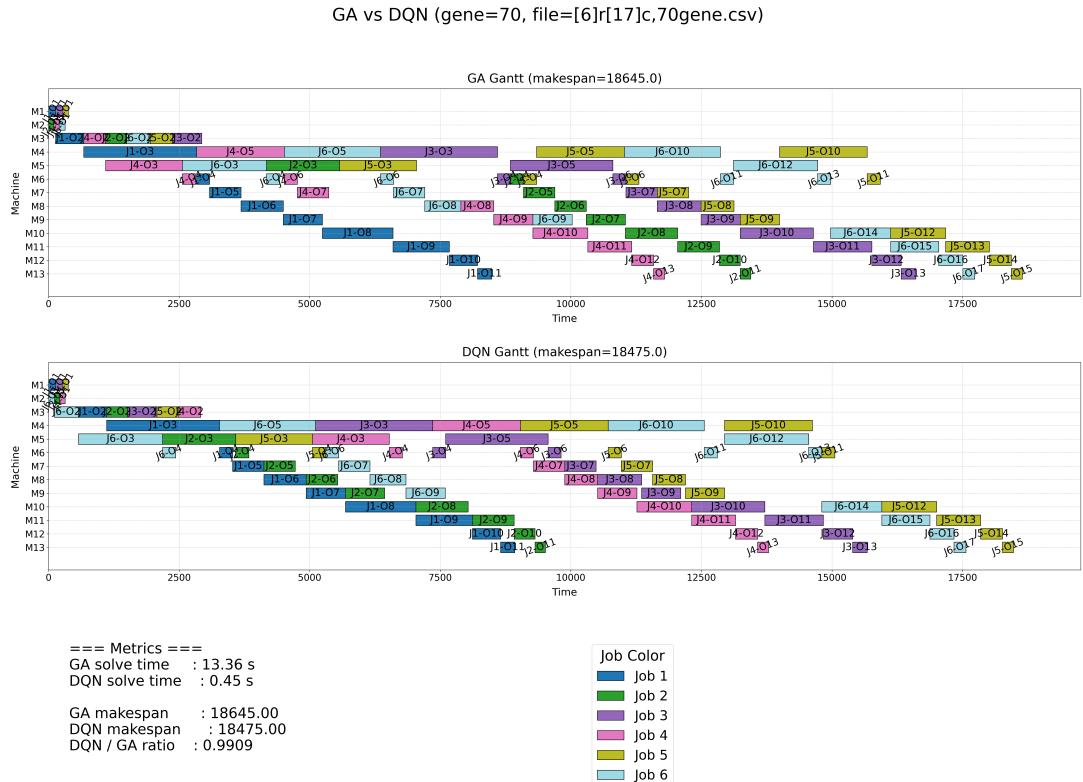


図 29: 注文(並)makespan 中心型報酬関数 DQN と GA のガントチャート(比率 0.99)

図30は注文(並)タイプに対して, makespan 中心型報酬関数 DQN で得られたガントチャートの makespan と GA で得られたガントチャートの makespan の比率が 1.2 の例. このジョブセットに対して,DQN で得られたスケジュール案のパターンは図 27 とてている. ジョブ 6 の取り扱いが苦手の原因で, makespan を伸びた.

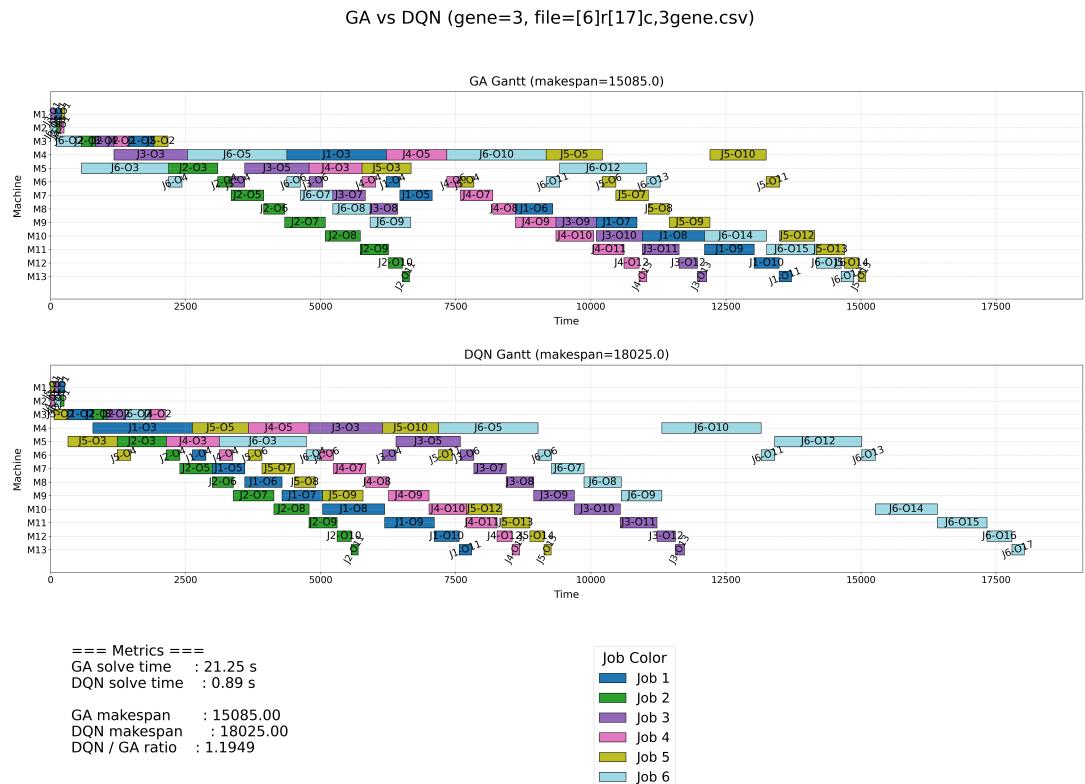


図 30: 注文(並)makespan 中心型報酬関数 DQN と GA のガントチャート(比率 1.2)

図31は注文(大)タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 0.94 の例. このジョブセットに対して,DQN で得られたスケジュール案のパターンは図 28 とてている. ジョブ 1 の取り扱いかたでスケジュール案を改善した.

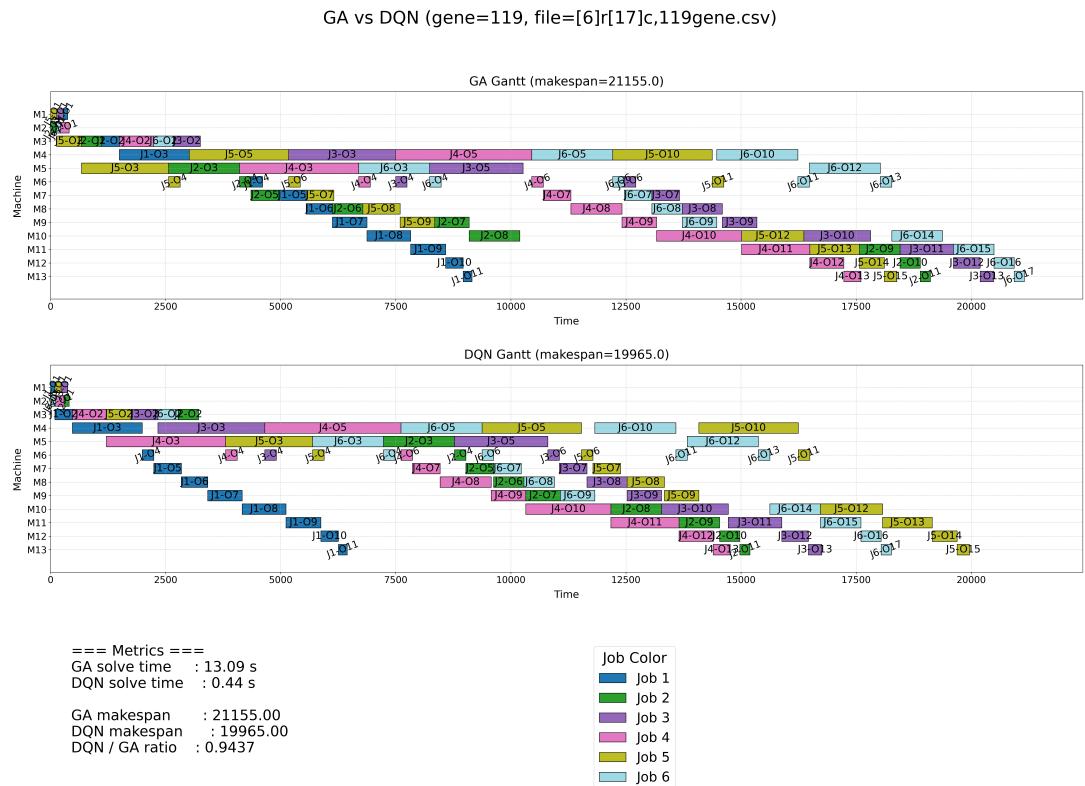


図 31: 注文(大)*makespan* 中心型報酬関数 DQN と GA のガントチャート (比率 0.94)

図32は注文(大)タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 1 の例. このジョブセットの二つのスケジュール案は *makespan* から見るとほぼ同じ程度である. 加工プロセスの塊は, GA の方は後で集中加工する.DQN の方は前置きに加工する方策を選択した.

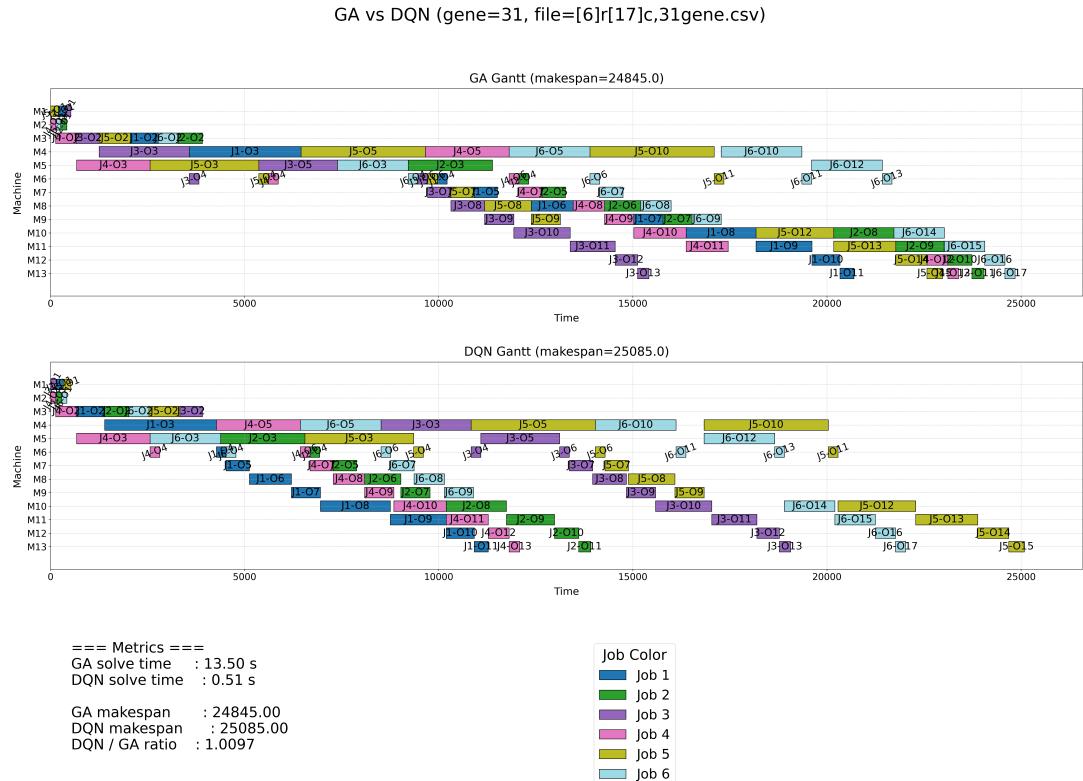


図 32: 注文(大)*makespan* 中心型報酬関数 DQN と GA のガントチャート(比率 1)

図33は注文(大)タイプに対して, *makespan* 中心型報酬関数 DQN で得られたガントチャートの *makespan* と GA で得られたガントチャートの *makespan* の比率が 1.16 の例. このジョブセットで DQN が得られたスケジュール案は前の比率が 1.2 弱の例と同じ, ジョブ 6 の加工方策は全ジョブの最後に配置し加工することを選択した. これにより *makespan* が GA と比べて劣れた.

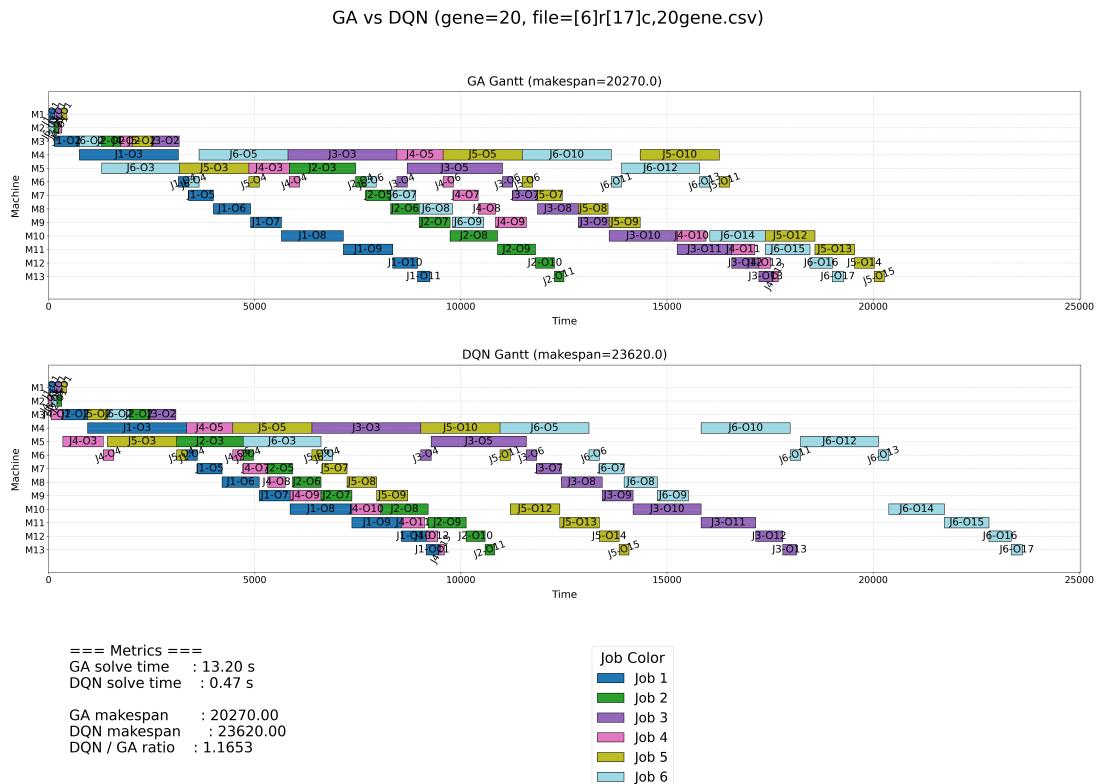


図 33: 注文(大)*makespan* 中心型報酬関数 DQN と GA のガントチャート(比率 1)

以上に示した図より,  $makespan$  中心型報酬関数を用いた場合, 多くの問題例においては, DQN がスケジューリング方策を適切に学習し, GA による解と同等, あるいはそれを上回る性能を示すことが確認された. これは, 報酬が  $C_{\max}$  の変化に直接基づいて与えられることで, ネットワークが調度全体の完工時間を意識した意思決定を獲得できたためであると考えられる.

図 34 には  $makespan$  を横軸, 縦軸はジョブセット番号順で並べた. ここではジョブセットに含まれる注文の処理時間の分散を 3 段階で色分けし, GA と DQN の  $makespan$  差を示す.

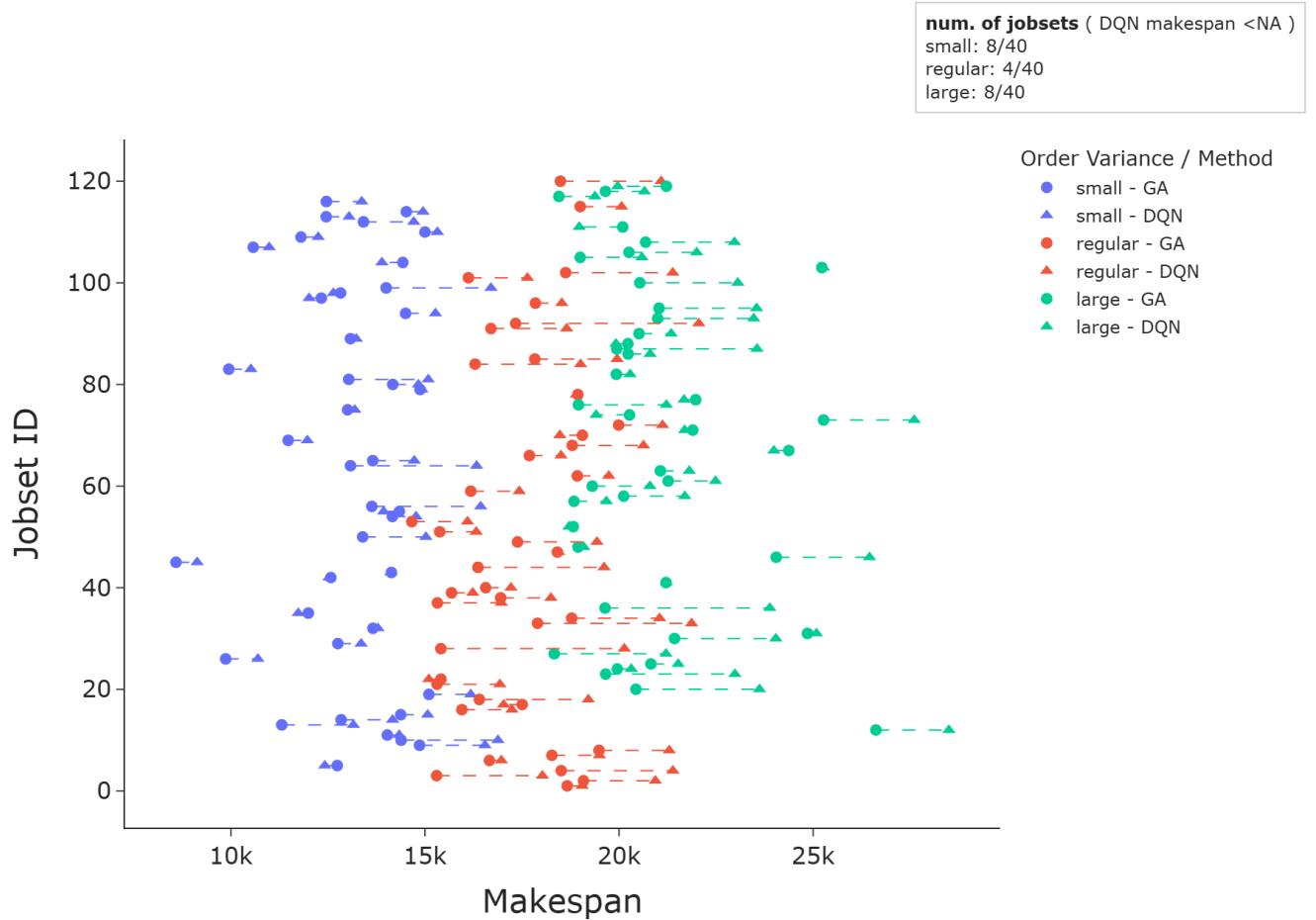


図 34: GA と  $makespan$  中心型報酬関数 DQN で得られたスケジュール案の  $makespan$  散布図

一方で, 一部の問題例においては, 加工工程数が最も多い Job6 がスケジュールの最後に集中して配置され, 結果として  $makespan$  が大きく悪化するケースも観測された. この挙動は,  $makespan$  中心型報酬関数における即時報酬と将来報酬の相対的な評価に起因すると考えられる.

すなわち, Job6 に対して他の作業を挿入する行動は, 短期的には即時報酬が小さく評価される一方で, 過去の選択によって見込まれる将来報酬が相対的に大きくなる場合がある. その結果, エー

---

ジェントは局所的に高い Q 値を持つ行動を優先し、加工工程数の多いジョブを後回しにする方策を選択する傾向を示す。

このように、*makespan* 中心型報酬関数は、全体として有効なスケジューリング戦略を学習させる能力を持つ一方で、報酬設計の性質上、特定のジョブ構成に対しては *makespan* の増大を招く意思決定が強化される可能性があることが示唆される。

## 6.6 実験考察

以上の実験結果より、本研究で検討した二種類の報酬関数は、いずれも DQN による JSSP スケジューリング方策の学習に寄与する一方で、異なる課題を含まれていることが明らかとなった。

まず、複合型報酬関数を用いた場合、平均 *makespan* や損失関数の推移から、エージェントが学習を進めているように見える挙動が確認された。しかしながら、報酬関数に含まれる要素が多岐にわたることで、各要素間の影響が相互に干渉し、学習信号にノイズが多く含まれる結果となつた。そのため、学習過程では一定の収束傾向が観測されるものの、最終的に得られた方策は実際の JSSP に対して十分に有効な解を導出できていない場合が多かった。すなわち、複合型報酬関数は「学習しているように見えるが、本質的な方策改善に結びつきにくい」という特性を有していると考えられる。

一方、*makespan* 中心型報酬関数を用いた場合、報酬が  $C_{\max}$  の変化に直接対応するため、エージェントはスケジューリング全体を意識した意思決定戦略を比較的明確に学習できる。その結果、多くの問題例において、GA と同等水準の解、あるいはそれを上回る解が得られた。しかし、即時報酬と将来報酬の評価バランスの影響により、加工工程数の多いジョブが後回しにされるなど、極端なスケジューリング方策が選択されるケースも確認された。これは、戦略そのものは学習できているものの、評価基準の歪みにより、特定条件下で性能劣化を引き起こす可能性を示している。

以上より、複合型報酬関数は安定した学習挙動を示しやすい反面、ノイズの多さにより有効な方策獲得が困難であり、*makespan* 中心型報酬関数は実用的な戦略を学習可能である一方、評価設計に起因する極端解の問題を内包していることが分かった。

---

## 7. 今後の課題

本研究を通じて,DQN を用いた JSSP スケジュール最適化の有効性と課題が明らかとなった. これらを踏まえ, 今後の課題として以下の二点が挙げられる.

第一に, 報酬関数設計の高度化が挙げられる. 本研究では, 複合型報酬関数と *makespan* 中心型報酬関数をそれぞれ独立に検討したが, 複合型は学習挙動が安定しているように見える一方で, 多様な評価要素が同時に作用することにより報酬のノイズが増大し, 必ずしも有効なスケジューリング方策の獲得に結びつかない場合があった. 一方, *makespan* 中心型報酬関数では, スケジュール方策そのものは比較的明確に学習されるものの, 即時報酬と将来報酬のバランスが偏ることで, 一部の作業を後回しにする極端な意思決定が選択される傾向が確認された. 特に, 加工工程数の多いジョブに対して, 他ジョブを挿入することによる即時的な報酬改善よりも, 過去の行動に基づく将来的報酬が過大に評価される場合, 結果として総 *makespan* を悪化させる調度が生成される可能性がある. 今後は, 即時報酬と将来報酬の寄与度を適切に調整する仕組みや, 学習段階に応じて報酬構造を変化させる手法を導入することで, GA と比較しても極端な解を生じにくく, より安定したスケジューリング方策の獲得が期待される.

第二に, モデルの一般化性能の向上が課題として挙げられる. 本研究では, 製造する製品の加工工程順序をあらかじめ固定した問題設定を採用した. そのため, 学習済みモデルは, 訓練時と同一または類似した加工工程構造を持つ問題に対しては有効である一方, 訓練時に出現しなかった加工工程順序を持つ製品に対する適応性には制限がある. 今後は, 加工工程順序が異なる製品を含む学習データの拡張や, 状態表現の改良を通じて, 未知の加工工程構造に対しても柔軟に対応可能な, より汎用的なスケジューリングモデルの構築が望まれる.

---

## 8. 謝辞

本研究を進めるにあたり、多大なるご指導とご支援を賜りました指導教員の先生に、心より感謝申し上げます。研究の遂行に際しては、論文執筆における日本語表現の細部に至るまで丁寧にご確認いただき、また、研究内容の構成や論理展開についても数多くの貴重な助言をいただきました。さらに、本課題を遂行する上で不可欠となる研究設備のご提供をはじめ、研究環境の整備においても多大なご配慮を賜りました。ここに深く感謝の意を表します。

---

## 参考文献

- [1] K.T. Chung, C.K.M.Lee and Y.P. Tsang, "Neural combinatorial optimization with reinforcement learning in industrial engineering: a survey" *Artif Intell Rev.* vol. 58, 130(2025).
- [2] M. Xu, Y. Mei, F.F. Zhang and M.J. Zhang, "Learn to optimise for job shop scheduling: a survey with comparison between genetic programming and reinforcement learning" *Artif Intell Rev.* vol. 58, 160(2025).
- [3] C. Ngwu, Y. Liu and R. Wu, "Reinforcement learning in dynamic job shop scheduling: a comprehensive review of AI-driven approaches in modern manufacturing" *J Intell Manuf.*(2025).
- [4] X.R. Shao and C.S. Kim, "An Adaptive Job Shop Scheduler Using Multilevel Convolutional Neural Network and Iterative Local Search" *IEEE Access.* vol. 10, pp. 88079-88092(2022).
- [5] L.B. Wang, X. Hu, Y. Wang, S. Xu, S. Ma, K. Yang, Z. Liu and W. Wang, "Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning" *Computer Networks.* vol. 190, 107937(2021).
- [6] S. Lee, Y. Cho and Y.H. Lee, "Injection Mold Production Sustainable Scheduling Using Deep Reinforcement Learning" *Sustainability.* vol. 12, 8718(2020).
- [7] C. Pickardt, J. Branke, T. Hildebrandt, J. Heger and B. Scholz-Reiter, "Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness" *Proceedings of the 2010 Winter Simulation Conference.* Baltimore, MD, USA, pp. 2504-2515(2010).
- [8] 村山 昇, 川田 誠一, "遺伝的アルゴリズムを用いた加工機械と複積載 AGV の同時スケジューリング" 日本機械学会論文集 (C編). vol. 12, 712, pp. 3638-3643(2005-12).
- [9] B. Yu, Z. Hui, Z. Xin, C. Zheng, J. Riku, Y. Kun, "Toward Autonomous Multi-UAV Wireless Network: A Survey of Reinforcement Learning-Based Approaches" *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS.* vol. 25, No. 4, FOURTH QUARTER 2023, pp. 3038-3067(2023).

- 
- [10] 小野 啓介, 森川 克己, 長沢 敬祐, 高橋 勝彦, ”フレキシブルジョブショップ環境の受託製造企業におけるエネルギー消費量配分問題” *J Jpn Ind Manage Assoc.* vol. 72, pp. 179-187(2022).
- [11] 貝原 俊也, 國領 大介, 藤井 信忠, 村上 亘, 梅田豊裕, ”フレキシブルジョブショップを対象とした受注生産における機械稼働計画立案もための基礎検討” 第 63 回自動制御連合講演会. (2020).
- [12] 貝原 俊也, 國領 大介, 藤井 信忠, 西村 翔平, ”CPS 型ファクトリセキュリティ実現に向けた生産スケジューリング手法に関する研究-マスカスタム生産対応フレキシブルオープンショップを対象とした検討-” 第 61 回自動制御連合講演会. (2018).
- [13] 平沼 智之, 安田 翔也, 藤堂 健世, 谷口 茉帆, 山村 雅幸, ”組合せ最適化におけるベイジアン最適化アルゴリズムを組み込んだ遺伝的アルゴリズムの提案” *The 35th Annual Conference of the Japanese Society for Artificial Intelligence.* (2021).
- [14] 三神 賢雅, 伊原 涼也, 佐久間 拓人, 加藤 昇平, ”混合整数最適化に基づく生産ライン作業スケジュール生成システムの開発” *The 36th Annual Conference of the Japanese Society for Artificial Intelligence.* (2022).
- [15] 蘭 嘉, 田中 瑛理, 佐々木 優, 森江 翔, 有馬 澄佳, ”複数種のリソースを共用する多品種生産システムの分散協調スケジューリング-自動車部品後補充生産への適用-” 日本経営工学会論文誌 . vol. 72, No.1, pp. 75-87(2021).
- [16] G.Y. Shi, H. IIMA, N. Sannomiya, ”A New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm” *Conference on Decision and Control.* (1996).
- [17] UMIT BILGE, GUNDUZ ULUSOY, ”A TIME WINDOW APPROACH TO SIMULTANEOUS SCHEDULING OF MACHINES AND MATERIAL HANDLING SYSTEM IN AN FMS” *Operations Research.* vol. 43, No.6, (1995).
- [18] 花田 良子, 廣安 知之, 三木 光範, ”遺伝的アルゴリズムによる工場の生産スケジュールの自動生成” *THE SCIENCE AND ENGINEERING REVIEW OF DOSHISHA UNIVERSITY.* vol. 48, No.4, pp.241-248, (2008).

- 
- [19] 平中 雄一朗, 西 竜志, 乾口 雅弘, ”ラグランジュ緩和とカット生成による生産工程と複数台搬送車の同時スケジューリング問題に対する分解法” システム制御情報学会論文誌.vol. 20, No.12, pp.465-474, (2007).
- [20] Raghda B. Taha, Amin K. El-Kharbotly, Yomna M. Sadek, Nahid H. Afia, ”A Genetic Algorithm for solving two-sided assembly line balancing problems” *Ain Shams Engineering Journal*.vol. 2, pp.227-240, (2011).
- [21] Kazi Shah Nawaz Ripon, N. H. Siddique, Jim Torresen, ”Improved precedence preservation crossover for multi-objective job shop scheduling problem” *Evolving Systems*.vol. 2, pp.119-129, (2011).
- [22] Andrzej Kiernich, Jerzy Kalenik, Wojciech Steplewski, Marek Koscielski and Aneta Chołaj, ”Impact of Particular Stages of the Manufacturing Process on the Reliability of Flexible Printed Circuits” *Sensors*.25, 140, (2025).
- [23] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, ”Human-level control through deep reinforcement learning”, *Nature*, vol. 518, no. 7540, pp. 529-533 (2015).
- [24] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, Cambridge, MA (2018).
- [25] M. Nazari, A. Oroojlooy, L. Snyder and M. Takáč, ”Reinforcement learning for solving the vehicle routing problem”, *arXiv preprint*, arXiv:1802.04240 (2018).
- [26] C. Zhang, W. Song, Z. Cao *et al.*, ”Learning to dispatch for job shop scheduling via deep reinforcement learning”, in *Advances in Neural Information Processing Systems* (NeurIPS) (2020).
- [27] D. Silver, J. Schrittwieser, K. Simonyan *et al.*, ”Mastering the game of Go without human knowledge”, *Nature*, vol. 550, pp. 354-359 (2017).

- 
- [28] J.N. Tsitsiklis and B. Van Roy,"An analysis of temporal-difference learning with function approximation", *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690 (1997).
- [29] 斎藤 康毅, 『ゼロから作る Deep Learning - Pythonで学ぶディープラーニングの理論と実装』, オライリー・ジャパン, 2022.
- [30] 野寺 隆志, 『楽々LATEX』, 共立出版株式会社, 1996.
- [31] 飯塚 修平, 『ウェブ最適化ではじめる機械学習』, オライリー・ジャパン, 2020.
- [32] 矢沢 久雄, 『基本情報技術者 らくらく突破 Python』, 技術評論社, 2021.
- [33] 伊藤 多一, 今津 儀充, 須藤 広大, など 『現場で使える! Python 深層強化学習入門 - 強化学習と深層学習による探索と制御』, 株式会社翔泳社, 2025.
- [34] Kirill Bobrov, 『なっとく! 並列処理プログラミング』, 株式会社翔泳社, 2025.
- [35] 平井 有三, 『はじめてのパターン認識』, 森北出版株式会社, 2023.
- [36] 大用 倉智, 山田 孝子, 『作りながら丁寧に学ぶ Python プログラミング入門』, 関西学院大学出版会, 2022.