強化学習

強化学習とは？

定義：「環境との相互作用を通じて,
　　試行錯誤しながら最適な行動を学ぶ枠組み」

他の機械学習との違い

- 教師あり学習：正解ラベルがある
- 教師なし学習：データの構造を探す

強化学習：行動の結果としての報酬だけを頼りに学ぶ

強化学習の成功例

産業・実務応用

自動運転

　車線変更やアクセル／ブレーキ操作の最適化に活用

　　（ただし実環境では安全性の課題あり）.

工場の生産計画やエネルギー最適化

　Googleがデータセンター冷却を強化学習で制御

　　→ 電力使用量を40%削減.

通信ネットワーク

　トラフィック制御やリソース割当で最適化に活用.

DeepMindはディープラーニング（深層学習）と強化学習を組み合わせた「Deep Reinforcement Learning（深層強化学習）」を開発しており、同技術は7月20日に囲碁の世界ランキング「Go Ratings」で世界一になった「AlphaGo」にも使用されている。今回Googleはこの技術を採用して、データセンター設備の稼働状態や気候などに応じて冷却設備の設定を最適化することで、冷却設備の消費電力を最小化するAIを開発した。

Googleはデータセンター内に数千個のセンサーを設置し、データセンターの各設備の温度や消費電力、ポンプの速度、各種設定といったデータを蓄積している（**写真**）。Googleは「ニューラルネットワーク」に、これらの稼働データと冷却設備の消費電力との間にあるパターンを学習させた。

写真●Googleのデータセンターにある冷却設備用の配管など

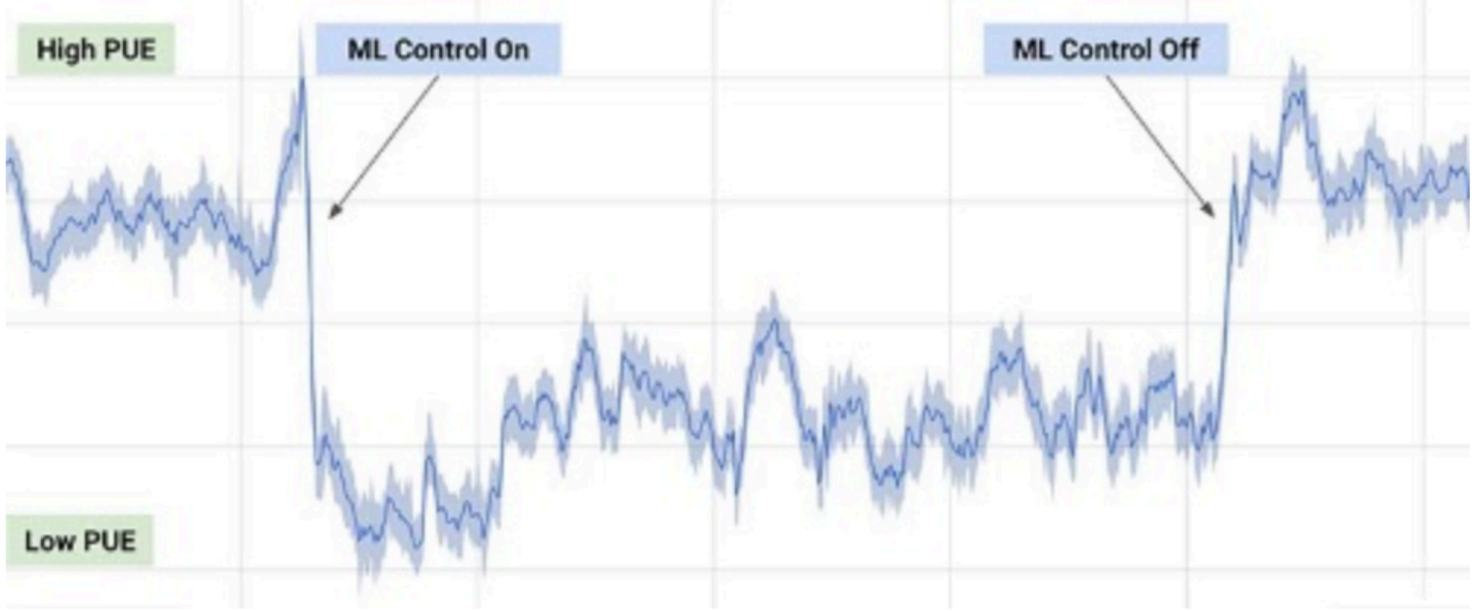
出典：米Google

[画像のクリックで拡大表示]

https://xtech.nikkei.com/it/atcl/news/16/072102162/

ニューラルネットワークはこうしたパターンを学習することで、データセンター内部や外部の環境に合わせて冷却設備の運用をどう変更すれば、冷却設備の消費電力を最小化できるか、冷却設備の「運用シナリオ」が分かるようになった。またニューラルネットワークにデータセンター周辺の気温と気圧の時系列データを学習させることで、1時間後のデータセンター周辺の気候を予測できるようになった。

Googleはこれらの分析結果を活用して、1時間後の気候や設備の稼働状況を予測した上で、その状況に最適な冷却設備の運用シナリオを適用するシステムを構築。予測に基づいて冷却設備の運用を細かく調整することによって、冷却設備の消費電力を最大40%削減できるようになったとしている（**図**）。



図●AIに運用を任せた場合のデータセンター電力消費効率

AIに運用を任せた「ML Control On」から「Off」までの間、電力消費効率が改善した（出典：米Google）

https://xtech.nikkei.com/it/atcl/news/16/072102162/

強化学習の基本要素

エージェント（学習者）：ロボット，プレイヤー

環境とやりとりしながら行動を選び，経験から学ぶ主体.

環境（問題の舞台）：迷路，ゲーム

エージェントが存在し，行動の結果や報酬を返す世界.

状態（環境の観測）：位置，センサー情報

ある時点で環境から得られる情報のスナップショット.

行動（エージェントの選択）：曲がる，止める，運ぶ

状態を見てエージェントが取りうる操作や決定.

報酬（良し悪しのフィードバック）：ゴール＋１，落下－１

行動の結果に対して環境から与えられる数値的な評価.

強化学習の考え方

目標：累積報酬の最大化
その場の得点だけでなく，将来まで含めたトータルの利益を大きくすること.

方策（policy）：行動を決めるルール
状態を見て「このときはこう動く」と選ぶための戦略や確率分布.

価値関数（value）：状態や行動の良さを数値化
その状態や行動を取ったときに，将来どれくらい報酬が期待できるかを表す指標.

強化学習の代表的な手法

価値ベース（例：Q-learning, DQN）
各行動の価値を推定し，最も価値の高い行動を選ぶ方法.

方策ベース（例：Policy Gradient）
方策そのものを直接調整して，良い行動を選ぶ確率を高める方法.

ハイブリッド（例：Actor-Critic）
「方策」で行動を選びつつ，「価値関数」でその評価を補助する組み合わせ手法（先のGoogleの手法）

Googleの冷却最適化では？

エージェント（学習者）：ロボット，プレイヤー

環境とやりとりし[illegible]冷却システムを操作するAI制御モデル

環境（問題の舞台）：迷路，ゲーム

エージェント[illegible]サーバー，冷却装置，外気温，電力系統の環境

状態（環境の観測）：位置，センサー情報

ある時点で環境から[illegible]センサーから得られる時系列データ

行動（エージェントの選択）：曲がる，止める，運ぶ

状態を見てエ[illegible]ポンプ速度，ファン回転数，冷却水量の調整

報酬（良し悪しのフィードバック）：ゴール＋１，落下－１

行動の結果に対して環境から与えられる数[illegible]電力効率を報酬

強化学習の考え方

迷路を解くロボット

ゴールに到達すれば+1，壁にぶつかれば−1

何度も試して，報酬の高い行動パターンを学ぶ

1．試行錯誤で経験をためる

迷路を解くロボットは，最初はランダムに動くので壁にぶつかる．でも「壁にぶつかる＝損」と学習すると，次は違う道を選ぶようになる.

2. 良い行動が“クセ”になる

ゴールに近づくたびに報酬が貯まるので，その行動パターンを「繰り返したくなる」．失敗は避けられ，成功の経験が強化される.

強化学習の考え方

迷路を解くロボット

ゴールに到達すれば+1，壁にぶつかれば−1

何度も試して，報酬の高い行動パターンを学ぶ

3. 未来を考えられるようになる

すぐの報酬よりも，将来のゴール（大きな報酬）を優先するようになる．その結果，「遠回りだけど確実にゴールできる道」を選べるようになる．

4. 記憶が「地図」になる

多くの試行を重ねるうちに，「この道はダメ」「この方向が良い」という地図のような知識ができる．これが「価値関数」や「方策」に相当する．

スケジューリング

スケジューリング：いろいろあります！

フローショップ・スケジューリング問題

すべてのジョブが同じ順序で機械を通過する生産ライン型のスケジューリング問題．課題は「ジョブを流す順番」を工夫して，全体の処理時間を最小化すること．

ジョブショップ・スケジューリング問題

ジョブごとに通過する機械の順序が異なる，より複雑なスケジューリング問題．課題は「どのジョブをどの機械にいつ割り当てるか」を最適化すること．

そもそもジョブとは？

ジョブ（Job）
完成させるべき「製品や注文」の単位.
例：自動車1台の製造，全工程を通して1つのジョブと考える.

タスク（Task / Operation）
ジョブを構成する個々の工程や作業単位.
例：自動車の製造なら「溶接」「塗装」「組立」などがタスク.

→ ジョブは複数のタスクから成る.

フロー（Flow）
ジョブが機械を通る順序（加工の流れ）を指す.

そもそもフローとは？

フロー（Flow）
ジョブが機械を通る順序（加工の流れ）を指す.

→ フローショップではすべてのジョブが同じ流れを通る.

ショップ（Shop）
工場や作業場のこと. ここで複数の機械や作業者がタスクを処理する.

→ ジョブショップは「工場に複数の機械があり，ジョブごとに通る順番が異なる」状況を指す.

ジョブショップ・スケジューリング問題とは？

ジョブショップ・スケジューリング問題（**JSP**; Job-shop Scheduling Problem）とは、順序関係のあるいくつかの作業を複数の機械で処理する場合に、評価指標（機械全体の稼働時間の最小化、作業の納期遅れの最小化など）を最適にするような機械の稼働スケジュールを決める問題である。

概要 [編集]

- いくつかの**仕事**と**機械**がある。
- ひとつの仕事は定められた順序（**技術的順序**）で各機械の処理を受けて完成に至る。技術的順序および各機械での処理時間は所与である。
- ある機械が同時に複数の仕事を処理することはできない。
- これらの制約のもと、すべての仕事が完了するまでの所要時間を最小にするよう、各機械でどの仕事をどんな順序で処理するかを決定する。
- 所要時間をメイクスパンと呼ぶ。

仕事と機械の数が大きくなると最適解を求めることが劇的に難しくなる。この問題は機械数が3以上のとき NP完全であることが知られている[1]。

https://ja.wikipedia.org/wiki/ジョブショップ・スケジューリング問題

ジョブショップ・スケジューリング問題とは？

ジョブごとに機械を通過する順序が異なる.

例：ジョブ1 → M2 → M1 → M3

ジョブ2 → M1 → M3 → M2 …

機械加工や部品製造など，複雑な工程に対応.

例：金属加工工場，半導体製造など.

ジョブごとに順序が異なる → 工場加工型 → より複雑で難しい

https://ja.wikipedia.org/wiki/ジョブショップ・スケジューリング問題

フローショップ・スケジューリング問題とは？

フローショップ・スケジューリング問題（FSP）は、n個の仕事とm個の機械がが全て機械1, …, 機械mの順番で処理する場面で、各機械は常に高々1つの仕事を処理し、また一度仕事を処理し始めたら、処理を中断する ことができない。また各仕事が各機械でかかる処理時間はそれぞれ異なる。このとき、処理の総所要時間や納期遅れ時間などの評価尺度が最適となるように仕事の処理順序を求める問題である。

FSPでは以下の仮定の下で仕事の処理順序を求める[3]。

- 全ての仕事は互いに影響を受けることなく処理を行うことができ、いつでも処理を開始することができる。
- 各機械は連続して利用可能である。
- 各機械は一度に高々 1 つの仕事を処理できる。
- 各仕事は一度にどこか 1 つの工程のみ割り当てることができる。
- 仕事の処理が開始されると、その処理は中断することができない。
- 仕事の順序に依存した段取り時間（各工程間に必要な準備時間）を考慮しない。

仕事数n、機械数mのとき、フローショップ・スケジューリング問題の実行可能スケジュール数（組み合わせ数）は $(n!)^m$ である。また各機械の仕事順序が全て同一の順列フローショップ・スケジューリング問題の実行可能スケジュール数は$n!$である[4]。仕事と機械の数が増大すると最適解を求めることが劇的に難しくなる。

https://ja.wikipedia.org/wiki/フローショップ・スケジューリング問題

フローショップ・スケジューリング問題とは？

フローショップ・スケジューリング問題（FSP）は、n個の仕事とm個の機械がが全て機械1, …, 機械mの順

すべてのジョブが同じ順序で機械を通過する.

例：ジョブ1 → M1 → M2 → M3,

ジョブ2 → M1 → M2 → M3 …

生産ライン型の工場（ベルトコンベアのように流れ作業）に対応.

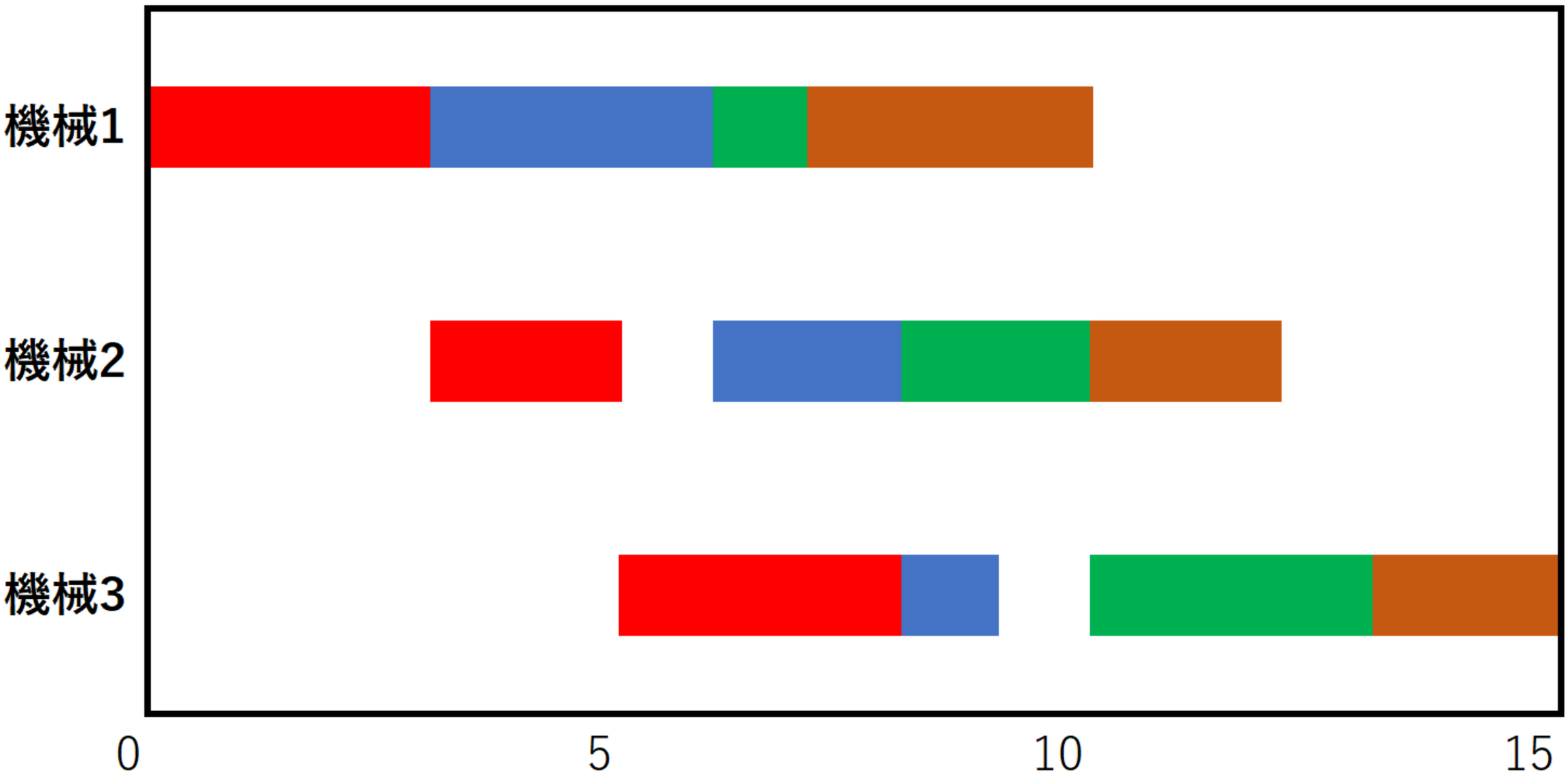
例：自動車の組み立てライン，食品工場など.

全ジョブが同じ順序 → 生産ライン型 → 比較的シンプル

る。

https://ja.wikipedia.org/wiki/フローショップ・スケジューリング問題

仕事数4, 機械数3のFSPに対するガントチャートの例



https://ja.wikipedia.org/wiki/フローショップ・スケジューリング問題

フローショップ問題：Johnson則

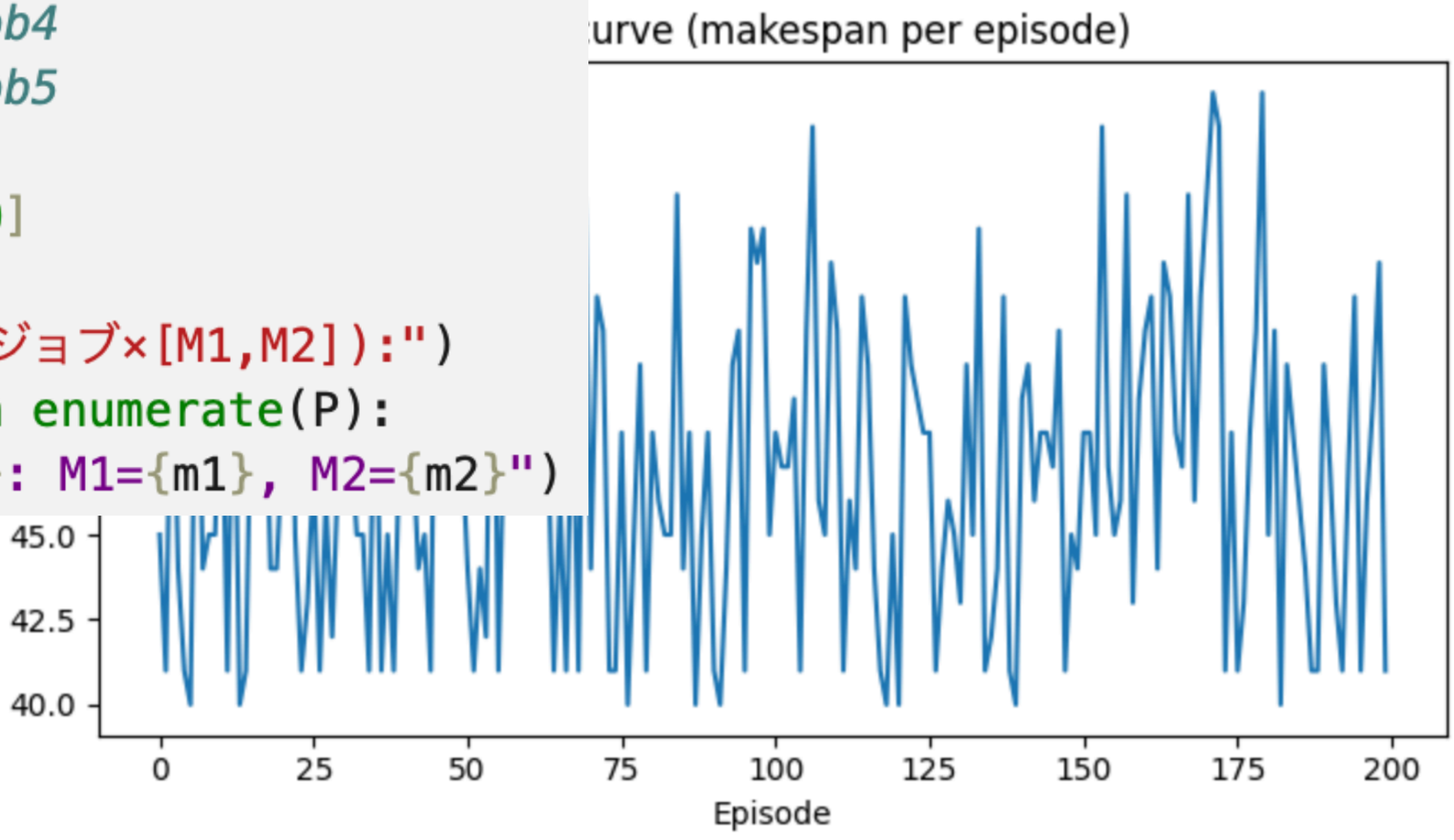
[4]:
```
# ===== 1) ジョブ処理時間を手動定義=====
# 各行が 1ジョブ: [M1時間, M2時間]
P = np.array([
    [3, 8],   # Job0
    [12, 2],  # Job1
    [5, 7],   # Job2
    [10, 3],  # Job3
    [2, 9],   # Job4
    [6, 6],   # Job5
], dtype=np.int32)
n_jobs = P.shape[0]

print("処理時間表 (ジョブ×[M1,M2]):")
for i, (m1, m2) in enumerate(P):
    print(f"Job{i}: M1={m1}, M2={m2}")
```

:
```
# ===== 実行例 =====
#  Johnson則
johnson_seq = johnson_order(P)
johnson_mk = makespan_of_order(P, johnson_seq)
print("\nJohnson則の順序:", johnson_seq, "→ makespan:", j
```

```
Johnson則の順序: [4, 0, 2, 5, 3, 1] → makespan: 40
```

フローショップ問題：強化学習

[4]:
```python
# ===== 1) ジョブ処理時間を手動定義=====
# 各行が 1ジョブ: [M1時間, M2時間]
P = np.array([
    [3, 8],   # Job0
    [12, 2],  # Job1
    [5, 7],   # Job2
    [10, 3],  # Job3
    [2, 9],   # Job4
    [6, 6],   # Job5
], dtype=np.int32)
n_jobs = P.shape[0]

print("処理時間表 (ジョブ×[M1,M2]):")
for i, (m1, m2) in enumerate(P):
    print(f"Job{i}: M1={m1}, M2={m2}")
```

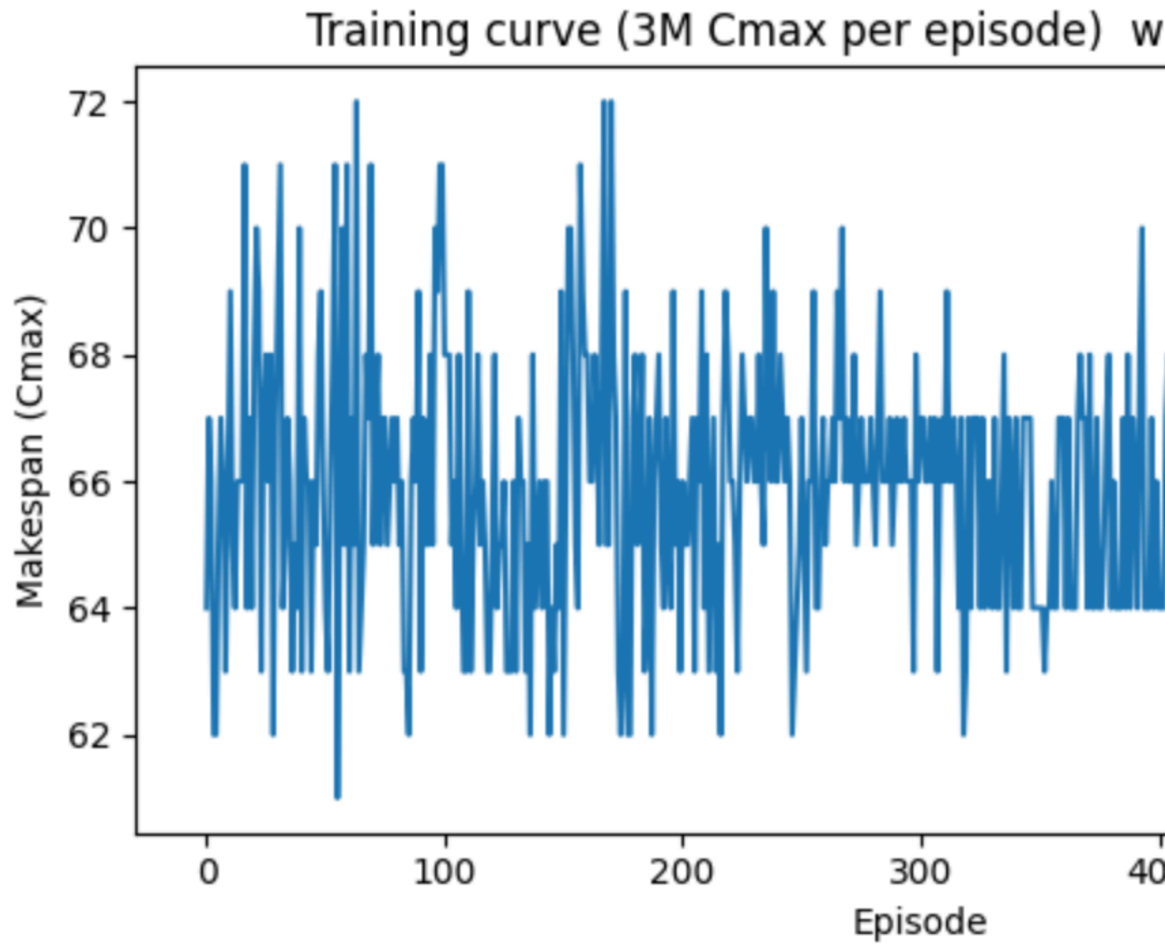



```
=== 強化学習の結果 ===
best sequence: [4, 5, 0, 2, 3, 1]
best makespan: 40
```

フローショップ問題　機械３台＋SPR：強化学習

```python
# ===== 1) ジョブの時間を手動定義（編集可）=====
# 各行が1ジョブ、列がそれぞれ [M1, M2, M3]
# P: 加工時間（process）、S: 段取り（setup）、R: 取り外し（removal）
P = np.array([
    [3,  6,  5],   # Job0
    [8,  2,  7],   # Job1
    [4,  5,  6],   # Job2
    [7,  3,  4],   # Job3
    [2,  9,  3],   # Job4
    [6,  4,  8],   # Job5
], dtype=np.int32)

S = np.array([
    [1, 1, 1],     # Job0 の段取り
    [2, 1, 1],     # Job1
    [1, 1, 2],     # Job2
    [1, 2, 1],     # Job3
    [1, 1, 1],     # Job4
    [2, 1, 2],     # Job5
], dtype=np.int32)

R = np.array([
    [1, 1, 1],     # Job0 の取り外し
    [1, 2, 1],     # Job1
    [1, 1, 1],     # Job2
    [1, 1, 2],     # Job3
    [1, 1, 1],     # Job4
    [1, 2, 1],     # Job5
], dtype=np.int32)
```

```
処理時間（process） P ：
Job0: 3,6,5
Job1: 8,2,7
Job2: 4,5,6
Job3: 7,3,4
Job4: 2,9,3
Job5: 6,4,8
段取り時間（setup）  S ：
Job0: 1,1,1
Job1: 2,1,1
Job2: 1,1,2
Job3: 1,2,1
Job4: 1,1,1
Job5: 2,1,2
取り外し時間（removal）R:
Job0: 1,1,1
Job1: 1,2,1
Job2: 1,1,1
Job3: 1,1,2
Job4: 1,1,1
Job5: 1,2,1
```




```
=== 強化学習の結果（3台 + 段取り・取り外し） ===
best sequence: [0, 2, 5, 1, 4, 3]
best makespan: 61
```

```python
# 好きな順序を評価：
my_order = [0,1,2,3,4,5]
print("my_order makespan:", makespan_order_3m_with_setup(P,
```

```
my_order makespan: 66
```

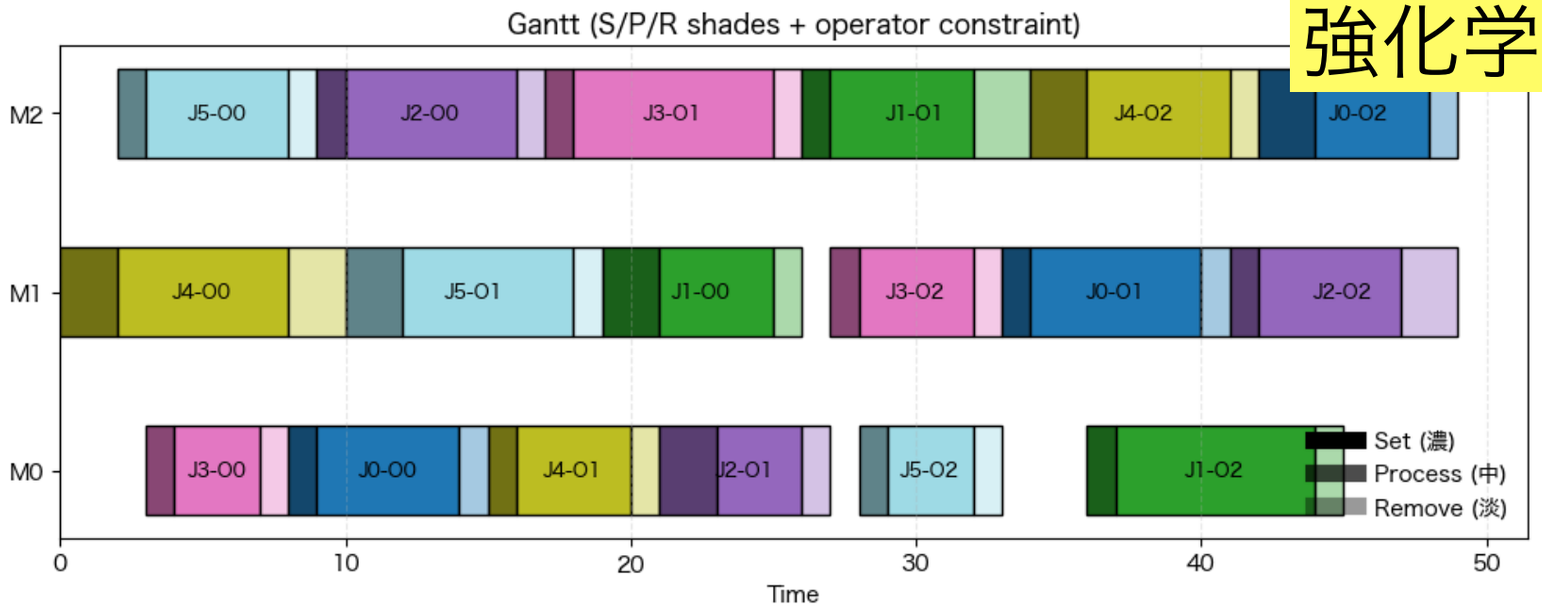
ジョブショップ問題　機械３台＋SPR：強化学習

```
# ===== 問題定義（編集可）=====
N_JOBS = 6
N_MACH = 3
N_OPS  = 3  # 各ジョブの工程数

# ジョブごとのルーティング（どの機械を使うか）
routes = np.array([
    [0, 1, 2],  # Job0: M0 -> M1 -> M2
    [1, 2, 0],  # Job1: M1 -> M2 -> M0
    [2, 0, 1],  # Job2: M2 -> M0 -> M1
    [0, 2, 1],  # Job3: M0 -> M2 -> M1
    [1, 0, 2],  # Job4: M1 -> M0 -> M2
    [2, 1, 0],  # Job5: M2 -> M1 -> M0
], dtype=np.int32)

# 時間（講義用に見やすい値。自由に編集OK）
S = np.array([
    [1, 1, 2],
    [2, 1, 1],
    [1, 2, 1],
    [1, 1, 1],
    [2, 1, 2],
    [1, 2, 1],
], dtype=np.int32)

P = np.array([
    [5, 6, 4],
    [4, 5, 7],
    [6, 3, 5],
    [3, 7, 4],
    [6, 4, 5],
    [5, 6, 3],
], dtype=np.int32)

R = np.array([
    [1, 1, 1],
    [1, 2, 1],
    [1, 1, 2],
    [1, 1, 1],
    [2, 1, 1],
    [1, 1, 1],
], dtype=np.int32)
```

```
=== Problem (3-machine JSP + S/P/R + 1-operator for Set) ===
Job0: route=(M0 -> M1 -> M2) | S=(1, 1, 2) P=(5, 6, 4) R=(1, 1, 1)
Job1: route=(M1 -> M2 -> M0) | S=(2, 1, 1) P=(4, 5, 7) R=(1, 2, 1)
Job2: route=(M2 -> M0 -> M1) | S=(1, 2, 1) P=(6, 3, 5) R=(1, 1, 2)
Job3: route=(M0 -> M2 -> M1) | S=(1, 1, 1) P=(3, 7, 4) R=(1, 1, 1)
Job4: route=(M1 -> M0 -> M2) | S=(2, 1, 2) P=(6, 4, 5) R=(2, 1, 1)
Job5: route=(M2 -> M1 -> M0) | S=(1, 2, 1) P=(5, 6, 3) R=(1, 1, 1)
```

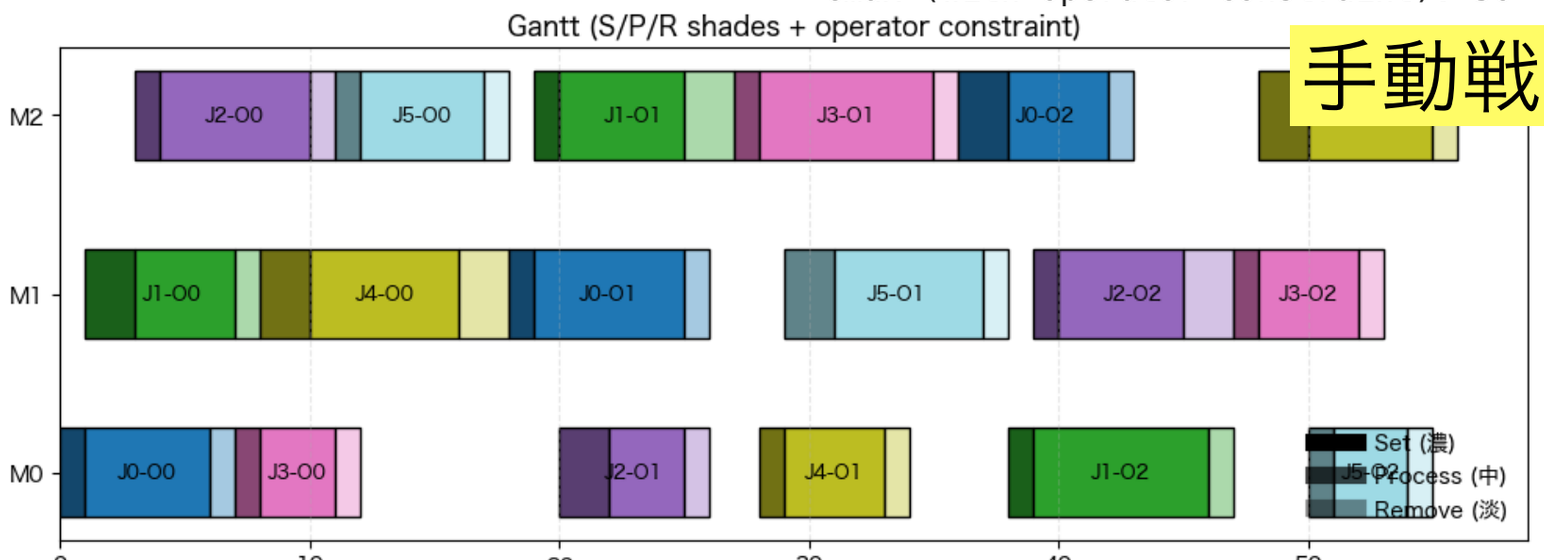
```
Cmax (with operator constraint): 49
```

強化学習



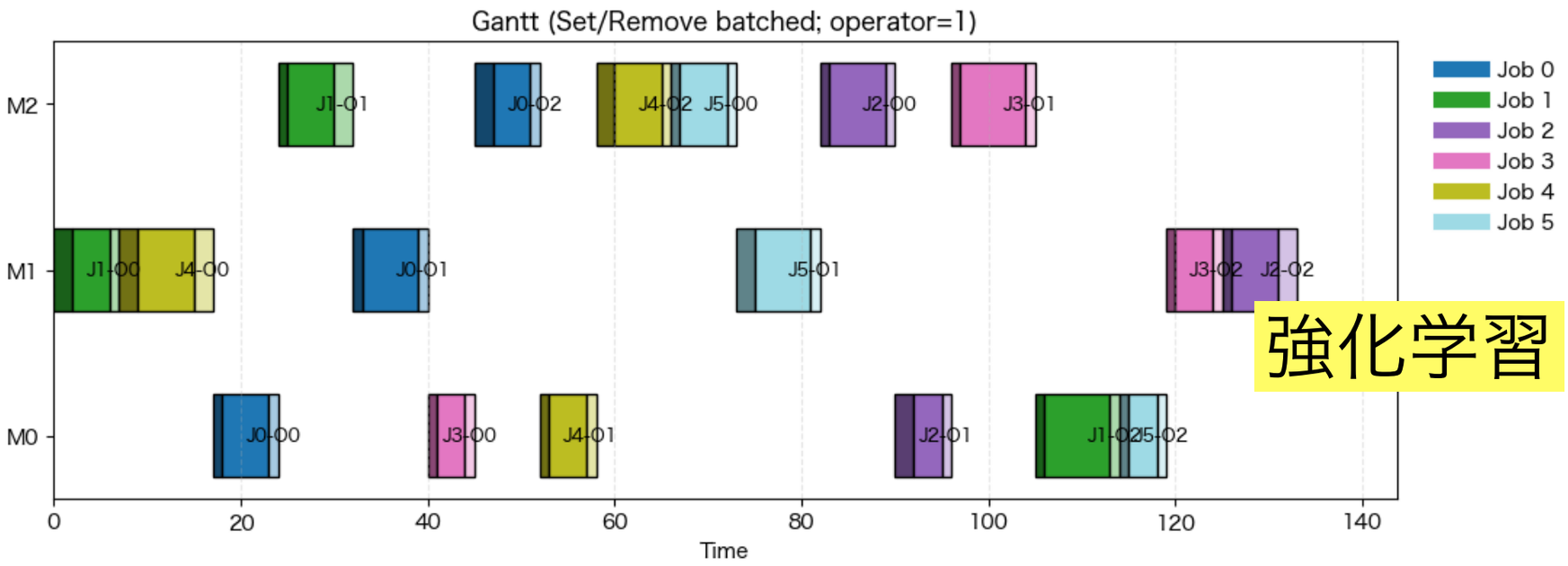

```
Cmax (with operator constraint): 56
```

手動戦略



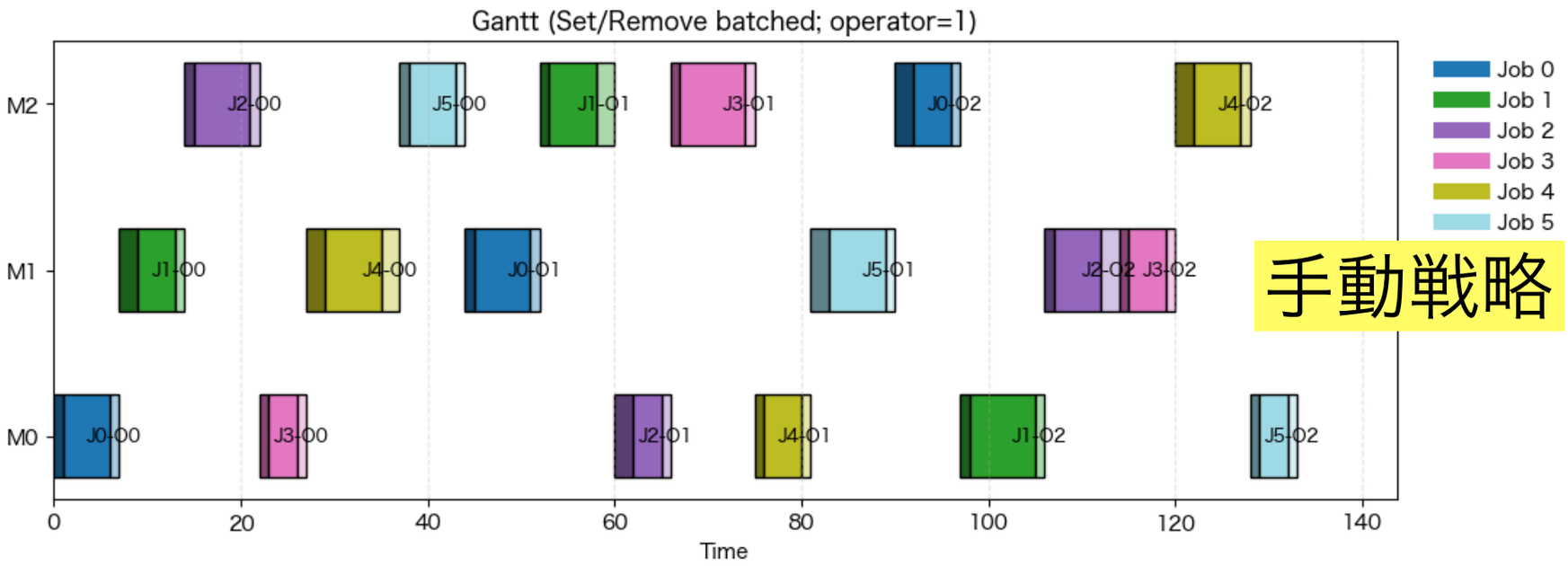
```python
routes = np.array([
    [0, 1, 2],  # Job0
    [1, 2, 0],  # Job1
    [2, 0, 1],  # Job2
    [0, 2, 1],  # Job3
    [1, 0, 2],  # Job4
    [2, 1, 0],  # Job5
], dtype=np.int32)

# Set 時間（スロット0／1 用）
S1 = np.array([
    [1, 1, 2],
    [2, 1, 1],
    [1, 2, 1],
    [1, 1, 1],
    [2, 1, 2],
    [1, 2, 1],
], dtype=np.int32)

S2 = np.array([
    [2, 1, 3],
    [3, 1, 2],
    [2, 3, 2],
    [2, 1, 2],
    [3, 2, 3],
    [2, 2, 2],
], dtype=np.int32)

# Process / Remove (P は 1件あたりの取り外し時間)
P = np.array(
    [5, 6, 4],
    [4, 5, 7],
    [6, 3, 5],
    [3, 7, 4],
    [6, 4, 5],
    [5, 6, 3],
], dtype=np.int32)

R = np.array([
    [1, 1, 1],
    [1, 2, 1],
    [1, 1, 2],
    [1, 1, 1],
    [2, 1, 1],
    [1, 1, 1],
], dtype=np.int32)

# 機械ごとの Set 同時実行キャパ（例：M1だけ2）
SET_CAP_MAP = {0: 1, 1: 2, 2: 1}
SET_CAP = np.array([SET_CAP_MAP.get(m, 1) for m
```

ジョブショップ問題　機械３台＋SPR
機械は２つセットできる条件：強化学習



強化学習

どちらもCmax (Set&Remove batched; operator=1): 133



手動戦略