Technical Specification for MPS Scheduler

Last Updated: 2025-03-09

This specification defines a reinforcement learning (RL) task using Deep Q-Networks (DQN) to optimize a scheduling problem with simulated data. The objective is to minimize the total time (makespan) required to complete all tasks, subject to given constraints.

1 Problem Description

We consider a scheduling problem involving a set of jobs (tasks) to be processed on a set of machines, where each job has compatibility constraints with specific machines, processing times, and additional constraints such as capacity, material availability, and deadlines. The goal is to assign jobs to machines over time to minimize the makespan, defined as the time from the start until all jobs are completed. Given the NP-hard nature of such scheduling problems, we employ DQN to approximate an optimal policy using simulated data.

2 Environment Definition

2.1 Sets and Parameters

- **Jobs**: $J = \{j_1, j_2, \ldots, j_n\}$, where n is the number of jobs.
  Each $j_i = (stat_i, begin_i, duration_i, ddl_i)$
  - $stat_i$ shows current status of job j_i.
  - $begin_i$ marks the date when we start to process this job j_i
  - $durantion_i$ indicates how long will it take to finish job j_i
  - ddl_i is the deadline of job j_i
- **Machines**: $M = \{m_1, m_2, \ldots, m_k\}$, where k is the number of machines.
  Each $m_i = (stat_i, job_i)$
  - job_i marks which job is assign to the machine m_i
  - $stat_i$ shows current status of machine m_i.
- **Compatibility & Processing Time**: We have a Compatibility Matrix $C \in \mathbb{R}^{n\times k}$. $c_{ij} = -1$ if machine m_j and job j_i is not compatible, else $c_{ij} > 0$ and represents the processing time for j_i on machine m_j.
- **Current Time**: At each state, we can also observe the current real-world time stamp τ. This time stamp is also used to determine a "final" state. When $\tau \geq \tau_f$, we consider we finish a process cycle, and all the machines will stop.
- **Constraints**:
  - **Capacity**: Each machine $m_i \in M$ can process at most one job at a time.
  - **Material**: Jobs may depend on material availability, implicitly affecting start times (assumed to be incorporated into the simulation).

2.2 Simulated Data

Simulated instances are generated with:

- Fixed n and m for a given training setup.
- Random draw the Compatibility Matrix C from pre-defined distribution (e.g., uniform over $[1, C_{\max}]$). Then randomly mask some value to -1.
- Additional constraints (e.g., capacity limits) sampled to reflect realistic MPS scenarios.

3 Markov Decision Process (MDP) Formulation

The RL task is modeled as an MDP (S, A, P, R, γ), where the agent interacts with the environment at discrete decision points to assign jobs to machines or wait, optimizing the cumulative reward to minimize the makespan.

3.1 State Space S

A state $s \in S$ at time t is represented as a tuple

$$s = (\tau, J, M, C),$$

where:

- **Current Time**: $\tau \in \mathbb{R}^+$, the elapsed time since the episode start.
- **Job Statuses**: $j_i = (stat_i, begin_i, duration_i, ddl_i)$:
  - $stat_i \in \{-1, 0, 1\}$, which represents "complete", "not start" and "running" respectively.
  - $ddl_i \in \mathbb{R}^+$ is given.
  - $begin_i = -1$ if not started yet, else it's positive.
  - $duration_i$ is calculated from C.
- **Machine Statuses**: $m_i = (stat_i, job_i)$:
  - job_i marks which job is assign to the machine m_i. $job_i = -1$ when the machine is idle.
  - $stat_i = 0$ if the machine is idle. $stat_i = t$ if busy, with value equal to the completion time of the job currently assigned to m_i.

State Representation for DQN:

- J: Encode to a vector of size $4n$ (4 states per job). Normalize *duration*, *ddl* and *begin*.
- M: Encode to a vector of size $2n$ (2 states per machine).
- Total state vector size: $4n + 2m$.

3.2 Action Space A

The action space is discrete:

- **Assignment Actions**: $\{(i, k) \mid i \in J,\ k \in M\}$, where (i, k) assigns job j_i to machine m_k.
- **Wait Action**: $\{\text{"wait"}\}$, which advances time without assignment.
- Total actions: $|A| = n \cdot m + 1$.
- **Valid Actions**: At state s, an action (i, k) is valid if:
  - $j_i[\text{"stat"}] = 0$ (job i is unassigned).
  - $M_k[\text{"status"}] = 0$ (machine k is idle).
  - $c_{ik} > 0$ (machine k is compatible with job i).

  The "wait" action is always valid.

3.3 Transition Function $P(s' \mid s, a)$

A state is observed we a specific job j_l is finished on machine m_n. Given state $s = (t, J, M, C)$ and action a:

- **Necessary update**:
  - Time τ = current time.
  - Update $j_l[\text{"stat"}] = -1$ (mark job j_l is completed).
  - Update $m_n[\text{"job"}] = -1$ (unload job from machine m_n).

  - Update $m_n[\text{“stat”}] = 0$ (mark m_n is idle).

  Then we try to take action.

- **If** $a = (i, k)$ **and** $c_{ik} > 0$:
  - Update $m_k[\text{“job”}] = i$ (machine k busy until completion).
  - Update $m_k[\text{“stat”}] = \tau + c_{ik}$ (machine k busy until completion).
  - Update $j_i[\text{“stat”}] = 1$ (job i in progress).
  - Update $j_i[\text{“begin”}] = \tau$ (job i in progress).
  - New state: $s' = (\tau', J', M', C)$, where t' is the next finish time.
  - Reward: $r = j_i[\text{“ddl”}] - j_i[\text{“duration”}] - j_i[\tau]$.
- **If** $a =$ **“wait”**:
  - Can always choose to wait:
    - New state: $s' = (\tau', J', M', C)$.
    - Reward: -1.

  Then, we have a “terminal state”

- **Terminal State**: If no machines are busy and all jobs are completed, transition to a terminal state, or time is up, $\tau > \tau_f$.
  - Stop.
  - Reward: $\sum \mathbb{I}(1 - j_i[\text{“stat”}])(j_i[\text{“ddl”}] - \tau)$.
- **Deterministic Transitions**: $P(s' \mid s, a) = 1$ for the resulting s' and 0 otherwise.

3.4 Reward Function $R(s, a, s')$

$$R(s,a,s') = \begin{cases} j_i[\text{“ddl”}] - j_i[\text{“duration”}] - j_i[\tau], & \text{for } a = (i,k), \\ -1, & \text{for “wait”}, \\ \sum \mathbb{I}(1 - j_i[\text{“stat”}])(j_i[\text{“ddl”}] - \tau), & \text{for terminal state,} \end{cases}$$

The objective is to maximize $\sum_t r_t$, which is equivalent to minimizing the total earliness of job completion.

3.5 Initial and Terminal States

- **Initial State**: $s_0 = (0, [0]^n, [0]^m, C)$ (time 0, all jobs unassigned, all machines idle).
- **Terminal State**: The episode ends when $J_{\text{status}}[i] = 1$ for all $i \in J$.

3.6 Discount Factor γ

$$\gamma = 1 \quad \text{(undiscounted)},$$

as the goal is to minimize total time without future discounting.

4 DQN Implementation

4.1 Q-Network Architecture

- **Input**: State vector of size $4n + 2m$.
- **Output**: Q-values for each action, of size $n \cdot m + 1$.
- **Layers**: A fully connected network, for example, with 2–3 hidden layers containing 128–256 units each and using ReLU activations.

4.2 Action Selection

- **Policy**: ϵ-greedy over $Q(s, a)$.
- **Invalid Actions**: For invalid (i, k), set $Q(s, (i, k)) = -\infty$ (masking) or treat as a “wait” action during execution.

4.3 Training

- **Experience Replay**: Store transitions (s, a, r, s') in a replay buffer.
- **Loss Function**: Minimize

$$L = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right],$$

  where θ^- denotes the target network parameters.
- **Optimizer**: Adam with a learning rate of approximately 0.001.
- **Simulated Data**: Generate episodes with varying p_{ik}, M_i, and constraints, training on thousands of instances.

5 Mathematical Objective

The agent learns a policy $\pi : S \to A$ to maximize the expected cumulative reward:

$$V_\pi(s_0) = \mathbb{E}_\pi\left[\sum_{t=0}^{T} r_t \mid s_0\right].$$

Given the reward function:

$$R(s, a, s') = \begin{cases} j_i[''ddl''] - j_i[''duration''] - j_i[\tau], & \text{for } a = (i, k), \\ -1, & \text{for ''wait''}, \\ \sum \mathbb{I}(1 - j_i[''stat''])(j_i[''ddl''] - \tau), & \text{for terminal state}, \end{cases}$$

our objective is to maximize the total earliness of job completion, defined as:

$$E = \sum_{i \in J} \left(j_i[''ddl''] - \text{completion time of } j_i\right),$$

where completion time of j_i is determined by the policy π.

Since the episode ends when all jobs are completed, the total reward is:

$$V_\pi(s_0) = \sum_{i \in J} (j_i[''ddl''] - j_i[\text{completion time}]).$$

This formulation is equivalent to minimizing the makespan $C_{\max}$, while also encouraging jobs to finish as early as possible before their deadlines. In other words, maximizing $V_\pi(s_0)$ prioritizes schedules that complete jobs sooner rather than later, leading to efficient job sequencing.