Enhancing Scalability in Distributed Flexible Flowshop Scheduling: A Hybrid RL-CP approach

Ioannis Avgerinos*, Christos Katrinakis*, Andreas Ktenidis*, Aggelos Ioannis Lagos*, Ioannis Mourtos*, Georgios Zois*,†

*ELTRUN Research Lab, Department of Management Science and Technology, Athens University of Economics and Business, Athens 104 34, Greece
†Optiscale, Athens 11472, Greece
iavgerinos, chr.katrinakis, and.ktenidis, t8210079, mourtos, georzois@aueb.gr

Abstract. Manufacturing-as-a-Service (MaaS) leverages decentralised resources provided by a network of manufacturers, to deliver on-demand production services to consumers. The core challenge lies in optimally allocating resources to service requests, balancing resource utilisation with consumer satisfaction. This context introduces large-scale, multi-objective optimisation problems that traditional exact methods, such as Constraint Programming (CP), struggle to solve efficiently. Motivated by a MaaS setting of two competing producers of electronic boards for white appliances, we propose a hybrid Reinforcement Learning (RL)-CP approach for the distributed hybrid flexible flowshop scheduling problem. We aim at minimising total earliness and tardiness, while ensuring fairness among the providers by minimising load imbalance. An RL agent divides the set of requests into smaller subsets to minimise load imbalance, while yielding smaller job sets solvable through a CP model to minimise total earliness-tardiness. To evaluate our hybrid RL–CP framework, we benchmark it against a relaxed CP model and a multi-phase constructive metaheuristic. As we show, the hybrid RL-CP outperforms both the CP model and the metaheuristic baselines in most instances, within practical computation times.

Keywords: distributed flowshop, constraint programming, reinforcement learning, manufacturing-as-a-service, large-scale

1 Introduction

Overview. MaaS enables the access to manufacturing capabilities on demand, often via digital platforms, and leverages distributed supplier networks, applying the "as-a-service" model to production. Production must be jointly orchestrated across multiple facilities in an efficient way that suits both the **providers** (who offer their manufacturing capacities to the network) and **consumers** (who submit manufacturing requests). Production-related constraints, particularly those

tied to the sequence of processes, create challenges for typical optimisation methods in scheduling. Although Constraint Programming (CP) has opened the door to richer models of these industry-specific conditions through interval variables, real-life instances involving thousands of jobs per week remains beyond the reach of exact CP methods. A common workaround is to decompose weekly workload into shorter planning horizons, hence obtaining subproblems manageable by CP. Yet these split-horizon decisions often lead to inefficient schedules, particularly in the presence of tight deadlines.

Problem description. We examine a MaaS paradigm motivated by a real industrial setting in which multiple competing manufacturers (*providers*) produce Electronic Boards (EB) for white appliances; each provider makes its production lines available for specific intervals. The resulting problem resembles a hybrid flexible flowshop scheduling (HFFS) setting: production lines follow a fixed sequence of stages that each job must traverse, stages may contain multiple parallel identical machines, and EB types may skip certain stages. Moreover, since no inter-industry transfers are allowed, the problem becomes a distributed HFFS (DHFFS). Each product is associated with a due-date, and deviations, whether early or late, incur penalties. This Just-In-Time (JIT) setting therefore calls for minimising the total earliness and tardiness across all products. The viability of MaaS depends on its trustworthiness from the providers' perspective, i.e., demand must be allocated as fairly as possible. This requirement introduces the additional objective of minimising the maximum load difference across providers.

Since no work-in-process storage is allowed in the examined problem, a product must move directly to the next stage as soon as it finishes the previous one. This restriction can introduce significant idle time: if no machine is available at the next stage, the product blocks its current machine until it can proceed. Transportation times between consecutive stages are assumed to be negligible.

Literature review. The DHFFS can be viewed as the culmination of a long line of scheduling models that progressively incorporate more realistic manufacturing conditions. Researchers introduced hybrid flowshops (HFS) to capture parallel identical machines at each stage [16]. The flexible extension of HFS captures that jobs may not need to be processed at all stages, but can skip some [12]. These conditions are further compounded by the need to assign jobs to distinct factories, each with its own hybrid flowshop structure. The DHFFS therefore requires three main decisions: assignment of jobs to factories, sequencing across stages, and machine selection per stage.

Scheduling problems in distributed hybrid flowshop settings are NP-hard, given that the hybrid flow shop is strongly NP-hard [13], and traditional exact methods (e.g., Integer Programming or CP) often struggle to scale [17]. Mixed-Integer Linear Programs (MILPs) for variations of the DHFFS have been proposed by [9, 22]. Also, [11] formulated three MILPs and a CP model for DHFFS with setup times to minimise the makespan. A state-of-the-art CP model for the HFFS with transportation times was proposed by [1], and it has largely inspired our own adaptation for the DHFFS.

Surveys of hybrid flowshops emphasise that high-performing metaheuristics combine constructive schedules based on dispatching rules with local search, iterated local search or iterated greedy [16]. For earliness/tardiness criteria, iterated local search and iterated greedy algorithms tailored to due-date–driven objectives have shown strong performance [14]. For distributed and blocking settings, [19] propose two problem-specific constructive heuristics, demonstrating that strong initial construction combined with targeted neighbourhood search substantially improves scalability.

Our standalone constructive metaheuristic follows this line of work: it generates diverse high-quality initial schedules using a portfolio of dispatching rules, and then applies large-scale neighbourhood search and simulated annealing tailored to the distributed, blocking DHFFS with fairness considerations. This provides a competitive, scalable non-learning baseline for large instances where standalone CP becomes impractical.

A recent systematic review by [15] highlights a rapid rise of RL methods for HFS, mostly in classical settings and predominantly using Q-learning or Deep Q-Network (DQN), with policy-gradient methods such as Proximal-Policy Optimisation (PPO) showing better scalability. In distributed settings, RL has predominantly been used to enhance metaheuristics for DHFFS variants. [21] integrate Q-learning into Teaching Learning-based Optimisation (TLBO) for a two-stage fuzzy DHFFS, while [4] employ Q-learning–guided local search within a conditional Markov Chain search. Energy- and priority-aware extensions also embed RL for operator control: [8] apply double DQN in a co-evolutionary algorithm, [25] incorporate Q-learning into metaheuristic hybrids with tailored local search, and [24] combine Particle Swarm Optimisation (PSO) with Q-learning-driven local search for bi-objective energy-efficient scheduling.

Recent studies have integrated RL with CP through several complementary paradigms. Some embed RL directly into CP solvers, where learned branching or value-selection heuristics operate alongside propagation and backtracking, as in SeaPearl and DRL-guided branch-and-bound for 3-D bin packing [3, 6]. A further line jointly formulates problems as RL environments and CP models, allowing learned policies to be integrated into CP search [2]. Finally, end-to-end scheduling approaches employ RL to learn dispatching rules through CP-backed simulators, achieving competitive job-shop performance [20]. To date, such RL–CP hybrids have not been explored for the DHFFS.

Contribution. We start by proposing a relaxed CP model for DHFFS, with the goal only to minimise total earliness and tardiness. Our main contribution is a RL-guided decomposition of the problem into two sequential decisions: (a) partitioning the dataset into smaller sequential instances, and (b) assigning each EB to one of the available providers. These decisions, made by a RL agent, decompose the original instance into a series of smaller and easier subproblems that can be efficiently solved by a restricted version of our CP within short time limits, ultimately producing complete high-quality solutions. To evaluate our hybrid RL-CP mechanism, we generate instances (from 500-10000 jobs) grounded in real-world data and expert knowledge, and establish baseline values from our

relaxed CP as well as from a state-of-the-art constructive metaheuristic, which builds full hybrid flow-shop schedules, combining dispatching rules, large-scale neighbourhood search and simulated annealing. Our hybrid RL-CP surpasses both CP and heuristic baselines for workloads of up to 2,000 jobs. As instance sizes increase, its performance remains stable, delivering solutions that are systematically superior in terms of earliness and tardiness while simultaneously achieving low load imbalance, all achieved within practical computational times.

Section 2 introduces the CP model for the problem under study. Section 3 presents the hybrid RL–CP framework, which is the main contribution of this work, followed by a summary of the constructive metaheuristic, which serves as a baseline for large-scale datasets. Section 4 provides numerical evidence of the framework's effectiveness compared with the standalone CP and metaheuristic components. Finally, Section 5 summarises the key findings of the study.

2 CP modelling

Problem formulation. Let J denote the set of jobs representing EB demands. Each job $j \in J$ has a due-date d_j; completing j one time unit early or late incurs an equal penalty. Let F denote the set of service providers, each consisting of a production line with stages S. A job j requires processing on a subset of stages $S_j \subseteq S$, which must be visited sequentially. To explicitly represent the processing order of stages, we denote each subset S_j as $\{s_j^1, s_j^2, ..., s_j^{|S_j|}\}$, where the superscript indicates the position of the stage in the sequence required by job j. Let M be the set of all machines, and let $M_{s,f} \subset M$ denote the machines in provider f dedicated to stage s. Processing job j at stage s in provider f requires at least $p_{j,s,f}$ minutes. H stands for the planning horizon.

CP model for JIT scheduling. We combine modelling components of state-of-art CP formulations for the HFFS problem to construct a model which captures all constraints of the problem as described above. For this model, we use a set of interval variables $\mathtt{x}_{j,s}$, indicating the processing of job j at stage s. Variables $\mathtt{providerOf}_j$ denote the provider which handles job j, thus their domain is set F. Optional interval variables $\mathtt{y}_{j,s,m}$ are synchronised with the respective variables $\mathtt{x}_{j,s}$ for exactly one machine $m \in M_{s,f}$, f being the value of variable $\mathtt{providerOf}_j$. Last, variables $\mathtt{E}_j$ and $\mathtt{T}_j$ represent the earliness and tardiness of job j.

$$\min \sum_{j \in J} \mathtt{E}_j + \mathtt{T}_j \tag{1}$$

$$\mathtt{alternative}(\mathtt{x}_{j,s}, [\mathtt{y}_{j,s,m} | f \in F, m \in M_{s,f}])) \qquad \forall j \in J, s \in S_j \tag{2}$$

$$\mathtt{startAtEnd}(\mathtt{x}_{j,s_j^i}, \mathtt{x}_{j,s_j^{i-1}}) \qquad \forall j \in J, i \in [2, |S_j|] \tag{3}$$

$$\mathtt{noOverlap}([\mathtt{y}_{j,s,m} | j \in J, s \in S_j \text{ if } m \in M_{s,f}]) \qquad \forall f \in F, m \in M \tag{4}$$

$$\mathtt{E}_j = \max(0, d_j - \mathtt{endOf}(\mathtt{x}_{j,s^{|S_j|}_j})) \qquad \forall j \in J \tag{5}$$

$$\mathtt{T}_j = \max(0, \mathtt{endOf}(\mathtt{x}_{j,s^{|S_j|}_j}) - d_j) \qquad \forall j \in J \tag{6}$$

$$\text{if } \mathtt{providerOf}_j = f \rightarrow \sum_{m \in M_{s,f}} \mathtt{presenceOf}(\mathtt{y}_{j,s,m}) = 1 \qquad \forall j \in J, s \in S_j, f \in F \tag{7}$$

$\mathtt{x}_{j,s}$: interval variable — $\forall j \in J, s \in S_j$

$\mathtt{y}_{j,s,m}$: interval variable

$\quad \mathtt{size} \in [p_{j,s,f}, H], \mathtt{optional}$ — $\forall j \in J, s \in S_j, f \in F, m \in M_{s,f}$

$\mathtt{providerOf}_j \in F$ — $\forall j \in J$

The objective function (1) minimises the total earliness and tardiness. Constraints (2) synchronise the time intervals of $\mathtt{x}_{j,s}$ and $\mathtt{y}_{j,s,m}$ only for one eligible machine, which is also the one which processes stage s of job j. The succession of stages is ensured by (3): each stage of a job starts at the end of the previous one. The `noOverlap` predicate of (4) prevents any processes from being simultaneously scheduled at the same machine. The values of earliness and tardiness are set by (5) and (6), given that $\mathtt{endOf}(\mathtt{x}_{j,s_j^{|S_j|}})$ denotes the end of the last stage of job j (i.e., the completion time of j). Finally, (7) ensure that each stage s of a job j is assigned to one of the machines of the selected provider, as indicated by the value of variable $\mathtt{providerOf}_j$.

Notably, the CP model addresses only the JIT objective. It is possible to incorporate a load-imbalance objective, defined as the maximum difference between the total processing time of the demand allocated to any two providers. However, incorporating this objective has been observed to significantly degrade the model's performance, limiting its scalability to only a few hundred jobs. For this reason, the experiments in this work consider minimisation of total earliness and tardiness as the sole objective of the CP model.

Reduced CP model. Datasets containing thousands of EBs cannot be directly handled efficiently by the CP model described above. To address this, we assume that our learning mechanism provides subsets of jobs $\bar{J}^i \subset J$, where i indicates the scheduling order. Additionally, for each job j, the assigned provider $f_j \in F$ is predetermined. These prior decisions simplify the CP model significantly.

Specifically, all variables and constraints that previously involved the entire set of jobs J are now restricted to the subset $\bar{J}^i$. Furthermore, the variables $\mathtt{providerOf}_j$ are no longer necessary, as the provider assignments are already determined by the learning mechanism. Consequently, constraints (7) are eliminated, and constraints (2) are replaced with:

$$\mathtt{alternative}(\mathtt{x}_{j,s}, [\mathtt{y}_{j,s,m} | m \in M_{s,f^j}])) \qquad \forall j \in \bar{J}, s \in S_j \tag{8}$$

Additionally, the scheduling of $\bar{J}^i$ depends on the decisions made during the scheduling of $\bar{J}^{i\text{-}1}$. Since some machines are partially occupied by previously scheduled jobs, the minimum start times for these machines must be updated. Let $u_m^{i\text{-}1} = \max(\mathtt{endOf}(\mathtt{y}_{j,s,m}) | j \in \bar{J}^{i\text{-}1}, s \in S_j)$ denote the maximum completion time of machine $m \in M$ after scheduling $\bar{J}^{i\text{-}1}$. Then, for the successive subset $\bar{J}^i$, the start times of all present variables $\mathtt{y}_{j,s,m}$ must be greater/equal than $u_m^{i\text{-}1}$. Clearly, the quality of the final schedule, that is, the combined schedule across all subsets $\bar{J}^i$, depends heavily on the appropriate selection of job subsets and their assignments to providers.

3 A hybrid RL-CP approach

The decomposition of J into EB subsets $\bar{J}^i$ and the assignment of each job j to a provider f_j are produced by a learning mechanism that interacts repeatedly with the environment. In line with recent work on deep reinforcement learning and neural combinatorial optimisation for complex shop scheduling problems [10, 18, 23], we use a RL policy to guide the higher-level orchestration, while leaving the exact scheduling to a CP engine.

At a high level, our approach intertwines a RL decision layer with the CP scheduling layer. At first, the RL policy observes an aggregated state of the network, including information on backlog jobs, current provider workloads and projected completion times. Subsequently, based on this state, the policy selects a subset of jobs $\bar{J}^i$ and assigns each of them to one of the providers in F. We refer to these subsets as *baskets* hereinafter. Conditioned on these decisions, the reduced CP model is instantiated and solved, producing detailed schedules that respect precedence constraints of the flowshop environment. The resulting schedules induce a cost in terms of total earliness/tardiness and load imbalance across providers. This cost is converted into a scalar reward signal, which is used to update the RL policy. Figure 1 depicts the interaction between the employed components. The figure illustrates the interaction loop of the hybrid RL–CP. At

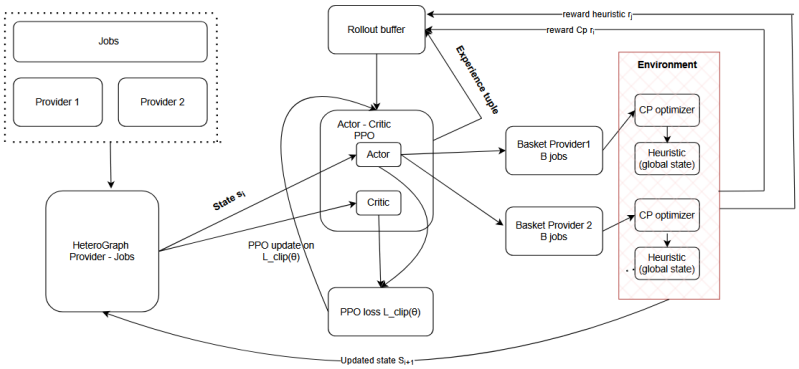

Fig. 1. A graph-based PPO allocates jobs to providers in baskets, CP Optimizer or the heuristic compute schedules and rewards, and PPO updates its policy via Lclip(θ)

each decision step, the current weekly instance (jobs and candidate providers) is embedded as a heterogeneous job–provider graph, which forms the state s_i. This graph is processed by the PPO actor–critic network: the actor outputs basket-level assignments of jobs to providers (e.g., baskets of size B for Provider 1 and Provider 2), while the critic estimates the state value $V_\theta(s_i)$. The selected baskets are then passed to the environment, where a CP Optimizer model constructs detailed hybrid flowshop schedules for each provider. A heuristic layer maintains a lightweight approximate global schedule between CP calls. Its role is to complement the basket-level reward from the CP solver by estimating how the agent's

latest assignment influences the estimated final global earliness/tardiness objective and the resulting load imbalance. After each basket is scheduled by CP, the heuristic updates these quantities by summing the processing times allocated to each provider, extending projected completion times accordingly, and estimating earliness/tardiness for unscheduled jobs via a simple earliest-finish prediction. This provides a global reward component that the CP solver alone cannot supply. We also experimented with more sophisticated constructive heuristics, such as the Phase-1 scheduler used in our standalone baseline, but these richer approximations led to worse policies. The resulting experience tuples (s_i, a_i, r_i, s_{i+1}) are stored in a rollout buffer and used to compute the PPO loss $L_{\text{clip}}(\theta)$, which updates the policy parameters. The updated state s_{i+1} is then re-encoded as a new job–provider graph, closing the loop for the next decision step.

In contrast to classical approaches, where batching and provider assignment are governed by hand-crafted heuristics, the RL agent *learns* over time which patterns of decomposition and allocation lead to subproblems which can be handled more easily by the CP optimizer, thus implying better performance overall. Importantly, the CP solver remains fully responsible for detailed scheduling: the learning component never replaces, but rather *steers*, the exact optimisation engine. Unlike prior RL-based methods that directly construct machine-level schedules for (flexible) job shops [10, 23], our agent operates at a higher level of abstraction, learning only the decomposition and provider-assignment policy while delegating fine-grained sequencing to CP.

3.1 MDP formulation and environment

We cast the sequential selection and assignment of subsets $\bar{J}^i$ as a Markov Decision Process (MDP). One episode corresponds to an instance with job set J and time horizon H, decomposed into decision steps $i = 1, \ldots, I$, where each step schedules one basket.

Operationally, the RL layer interacts with a simulation environment that emulates a planning horizon. Jobs are known upfront, but are revealed to the agent through a sequence of decision steps. At each step i, the environment provides a compact representation of (a) the set of unscheduled jobs, including due dates, processing times $p_{j,s,f}$, and stage masks S_j, (b) the current and projected workloads of each provider (accumulated processing time and estimated completion horizon), and (c) heuristic estimates of global earliness/tardiness.

State. We aggregate this information into a bipartite graph between jobs and providers. Job nodes carry features such as processing times, due-date, and stage-visit pattern, while provider nodes encode current load and projected completion time. This graph constitutes the state $\mathtt{s}_i$ fed to the policy, in a spirit similar to recent graph-based architectures for flexible job-shop scheduling [18, 23].

Action. Given state $\mathtt{s}_i$, the RL agent (i) selects a subset of jobs $\bar{J}^i \subset J$ to be scheduled next, and (ii) assigns each $j \in \bar{J}^i$ to a provider $f_j \in F$. To make this combinatorial decision tractable, we fix a maximum basket size B and parameterise the action as a matrix of scores $q_{j,f}$ over job-provider pairs. For each remaining job j, we identify the provider f with the highest score $q_{j,f}$, and use this

maximum score to rank jobs. The top-B jobs under this ranking form the basket $\bar{J}^i$, and their corresponding best providers define the assignment $\{f_j : j \in \bar{J}^i\}$.
CP-in-the-loop transition. Once $(\bar{J}^i, f_j, j \in \bar{J}^i)$ are chosen, they are passed to the reduced CP model. CP schedules $\bar{J}^i$ on the machines and stages of the selected providers, considering the precedence constraints between stages, and the inherited machine occupancy from previously scheduled subsets $\bar{J}^1, \ldots, \bar{J}^{i-1}$ through updated release times. The CP solver returns completion times and a detailed machine-level schedule, which are summarised by a heuristic layer into updated provider loads, projected completion times, load imbalance and global earliness and tardiness estimates, thus defining the next state $\mathtt{s}_{i+1}$.
Reward. The environment computes a scalar reward r_i that captures the quality of the CP schedule. The reward aggregates the total earliness/tardiness of $\bar{J}^i$ and a load-imbalance term reflecting the spread between provider workloads. A supplementary heuristic component (see Fig. 1) adjusts this reward by estimating the impact of the current assignment on the final global objective across all remaining jobs. Crucially, the RL agent does not modify the internal CP formulation: it influences only the sequence of baskets and their provider assignments.

3.2 Policy architecture and training

The policy is implemented using a graph-based actor-critic architecture trained with PPO. At each decision step, the bipartite job-provider graph is processed by a Graph Neural Network (GNN), which produces embeddings for both job and provider nodes. These embeddings are fed to an *actor* head, which outputs the job-provider score matrix $q_{j,f}$ from which baskets and assignments are derived, and a *critic* head, which estimates the value function $V(s_i)$ used for variance reduction in policy-gradient updates. This design is inspired by recent advances in GNN-based and attention-based architectures for job-shop and flexible job-shop scheduling [18, 23], but adapted here to operate at the level of provider assignment and instance decomposition rather than direct machine-level dispatching.

Training proceeds by simulating episodes on a set of instances of varying sizes. For each episode, the agent iteratively observes the current graph-encoded state $\mathtt{s}_i$, samples an action (basket and assignments) from the stochastic policy, and receives a reward obtained from the CP schedule and transitions to $\mathtt{s}_{i+1}$. Trajectories of states, actions, rewards and value estimates are stored and used to perform PPO updates on the policy and value networks. In our experiments, we train on a small, fixed collection of instances, cycling through one representative instance per job-size category at each training epoch, and performing a limited number of PPO epochs on the collected trajectories. Despite this modest training budget, the learned policy is able to discover allocation patterns that systematically improve downstream CP schedules compared to random or purely rule-based provider-assignment strategies, in line with empirical findings from related RL-based scheduling studies [10, 23].

3.3 (Meta)heuristic alternatives

A natural question is whether the decomposition and provider assignment decisions could be handled using alternatives methods. A first class of approaches would treat the assignment of jobs to providers, and possibly the partition of J into subsets $\bar{J}^i$, as a combinatorial optimisation problem on its own. Metaheuristics such as genetic algorithms, tabu search or simulated annealing could be designed to explore the space of assignment vectors f_j, or even full sequences of baskets $\{\bar{J}^1, \bar{J}^2, \dots\}$. Each candidate solution would then be evaluated by instantiating and solving the corresponding CP model, thus forming a classic *matheuristic* where CP acts as an embedded evaluation oracle.

While this strategy is conceptually appealing, it faces several practical challenges in the present MaaS context. At first, the number of possible provider assignments grows exponentially with $|J|$, and the number of ways to partition jobs into ordered subsets $\bar{J}^i$ is even larger. For instances containing thousands of jobs, exploring this space directly with a metaheuristic becomes extremely costly, as each evaluation requires solving multiple CP subproblems. Also, metaheuristics would typically treat each weekly planning instance as a new optimisation problem. Even if the underlying demand patterns and provider characteristics are similar from instance to instance, there is no built-in mechanism to reuse information learned on the past; the search restarts essentially from scratch. Crucially, the quality of the final global schedule depends not only on which jobs are assigned to each provider, but also on when they are scheduled within the planning horizon, and how earlier decisions affect later CP subproblems. Encoding this long-term, sequential coupling within a static metaheuristic objective is non-trivial.

In principle, more sophisticated matheuristics could be designed to address some of these issues; for example, by iteratively refining the assignment and reoptimising batches via CP, or by employing large neighbourhood search moves that reassign subsets of jobs between providers. However, these approaches still rely on manually crafted neighbourhoods and hand-tuned parameters, and they do not explicitly exploit the repetitive nature of the planning process (e.g., weekly rolling horizons with similar characteristics).

By contrast, the proposed RL layer is explicitly designed to treat successive instances as episodes drawn from a common distribution, allowing the policy to generalise across similar demand profiles and provider configurations, optimise decisions in a multi-step setting, where the consequences of allocating a job to a provider at time i are evaluated over the entire trajectory of subsequent CP schedules, thus reducing the amount of problem-specific algorithm design. In this sense, the RL-CP framework can be viewed as a *data-driven matheuristic*: CP remains the core optimisation engine for each subproblem, while the role traditionally played by a metaheuristic (navigating the space of high-level decisions) is taken over by a learned policy.

A standalone metaheuristic. Given the challenges described above, we implement a metaheuristic to provide a strong non-learning baseline for large instances where the CP model may time out. This metaheuristic constructs a complete

flow-shop schedule without relying on the CP model and optimises the same objectives as the RL–CP framework; namely, the total earliness/tardiness of all jobs and the load imbalance between providers, although the latter is considered only during the initial construction phase.

The metaheuristic operates in three phases. **Phase 1** generates multiple complete schedules using a single constructive scheduler. Jobs are first ordered by non-decreasing due date and assigned to the provider that minimises a score combining total processing time and a small load-balancing penalty. For each provider, we vary the *priority order* in which jobs are released to the constructive scheduler by applying classic dispatching rules (EDD, slack, critical ratio, weighted EDD, ATC) solely for ranking. The multi-stage scheduler then builds full blocking-aware schedules respecting machine availability and component capacities. Additional initial solutions are obtained by combining random provider assignments with the best sequencing rule. The best schedule found in this phase, measured by total earliness/tardiness, seeds the next phase.

Phase 2 performs iterative improvement using several operators: (i) job reassignment moves, which move a high-cost job to a different provider and rebuild the global schedule; (ii) job-swap moves, which exchange the providers of two jobs and reschedule; (iii) critical-job rescheduling, which removes a small set of jobs with the highest earliness/tardiness and reinserts them; and (iv) destruction–reconstruction steps, which temporarily remove a random fraction of scheduled jobs and reconstruct their assignments and sequences. Each move is evaluated by rebuilding the schedule with the same constructive scheduler and recomputing the earliness/tardiness objective. Improving moves are accepted greedily; if no improvement is observed for a given number of iterations, the search is restarted from the best solution found so far.

Phase 3 applies a simulated-annealing procedure to escape local optima. The same neighbourhoods as in Phase 2 (job reassignment, job swaps and provider-specific reordering) are sampled at random, but worsening moves may be accepted with probability $\exp(-\Delta/T)$, where Δ is the increase in total earliness/tardiness and T is a temperature parameter that is gradually cooled.

The best schedule found across all phases is returned as the final solution.

4 Experimental evaluation

All experiments run on a server equipped with 32 AMD Ryzen Threadripper PRO 5955WX 16-core processors and 32 GB of RAM, running Ubuntu 22.04.5 LTS. The CP model is solved by *CPLEX 22.1.1* optimiser, using the DOCplex module. All coding for the CP, learning and metaheuristic components are implemented in *Python 3.10.12*.

4.1 Generation of instances

The two providers in F resemble two major EB manufacturers. For our study, we focus on 10 EB types that typically account for a substantial share of the

weekly demand and require a production line consisting of 10 stages, denoted by $\mathtt{s}_i$, $i \in \{1, ..., 10\}$.

To generate instances with diverse characteristics, we assume that each EB type skips a randomly selected set of 0–2 stages, ensuring that at least 8 stages are required for processing. Processing times do not exceed 3.7 minutes. To ensure that the domains of start/end times for the CP interval variables contain strictly integer values, we measure time in deciminutes; consequently, the maximum processing time becomes 37 deciminutes.

A nominal processing time for each EB type is then drawn uniformlly between 0 and 37. To capture realistic but not excessive variation across providers, we introduce a noise parameter drawn uniformly between -2 and 2. For each provider, this noise value is added to the nominal processing time to obtain the final processing time used in the experiments. For every stage at each provider, we define between 1 and 9 identical parallel machines.

We generate 720 training instances, with 8 instances for each problem size (100, 200, 500, 800, 1,000, 2,000, 5,000, and 10,000 jobs). For evaluation, we use 18 distinct instances with 500, 800, 1,000, 2,000, 5,000, and 10,000 jobs. Each job is randomly assigned to one of the ten EB types, thereby inheriting its corresponding parameter values (e.g., processing times, component requirements). The due-date of each job is selected uniformly at random between 0 and H (i.e., a large value used as planning horizon).

4.2 RL training and learning protocols

Training. The provider-assignment policy is trained with PPO on the synthetic instances described above. Each training epoch consists of 8 different episodes, one per instance size (100, 200, 500, 800, 1,000, 2,000, 5,000 and 10,000 jobs). We train for 90 epochs in total, so the agent observes 720 distinct training instances and executes 141,120 decision steps (baskets of 100 jobs). After each epoch, we perform three PPO passes over the collected trajectories using a minibatch size of 16. Unless otherwise stated, we use a learning rate of 10^{-4}, discount factor $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, entropy regularisation weight 0.01, value-loss weight 0.5, and gradient clipping at 0.2. Training is carried out on a single GPU-equipped machine and requires approximately five days of wall-clock time.

Learning and evaluation protocol. We adopt an on-policy RL scheme in which each episode corresponds to the full scheduling of a single instance over the weekly horizon. At every decision step, the agent observes the aggregated state (backlog jobs, provider/factory loads, projected completion times, component-related indicators), selects a basket of 100 jobs and a provider assignment, and then receives a scalar reward computed from the CP-derived schedule (earliness/tardiness, load imbalance and idle time). After each epoch, the current parameters of the actor–critic network are stored as a checkpoint and evaluated in a strictly offline manner on a disjoint pool of 18 evaluation instances that covers the same size profile (500, 800, 1,000, 2,000, 5,000 and 10,000 jobs). During evaluation, the learned policy is run in deterministic (greedy) mode, and its

performance (measured primarily by the total earliness/tardiness and secondarily by load imbalance) is compared against CP baselines that use non-learning provider-assignment rules. The best model is selected as the checkpoint that achieves the lowest mean (ET) per job on this evaluation pool. In addition, we monitor a normalised load-imbalance metric that quantifies the fairness of the workload distribution across factories; this metric is reported for the selected checkpoint but is not used as a model-selection criterion.

During evaluation, we run the learned provider–assignment policy in *deterministic* (greedy) mode: at each decision step we select the factory with the highest probability under the trained actor, without exploration noise. The downstream CP solver and the heuristic component are also deterministic for a fixed instance. As a result, repeated evaluations of the same instance with the same checkpoint always produce identical schedules (identical total earliness/tardiness, load imbalance, and solution times).

For completeness, we run three repetitions per test instance and checkpoint, but these repetitions coincide up to numerical noise. The empirical standard deviation of the performance metrics across repeats is therefore (almost) zero, and we report only the mean values, which are equal to the outcome of a single deterministic run. The repetitions are used solely as a sanity check for determinism, and do not affect the reported results.

4.3 Results

Baselines from the CP model. To establish baseline objective values for evaluating the hybrid RL–CP mechanism, the first set of experiments applies our CP model directly to the full problem. Each instance is given a 3,600-second time limit. Because the CP model struggles to provide strong bounds, the time limit is reached for all but the smallest instances, and no meaningful lower bounds can be reported. The CP model managed to compute solutions for datasets of 1,000 jobs at most; the larger scales could not be handled within the time limit of one hour.

To obtain baseline objective values for datasets of 2,000 jobs, we relax our CP model, replacing constraints (3) with:

$$\texttt{endBeforeStart}(\mathtt{x}_{j,s_j^{i-1}}, \mathtt{x}_{j,s_j}) \qquad \forall j \in J, i \in [2, |S_j|] \tag{9}$$

In practice, substituting `startAtEnd` with `endBeforeStart` permits more flexible scheduling among the production flows of each job. This modification allows a machine to be released as soon as a stage is completed, even if the job is not immediately processed at the subsequent stage. Although the resulting schedules are not feasible within the actual industrial environment, the relaxed model is sufficiently accurate to serve as a reliable baseline for evaluating the hybrid RL–CP framework. For the 5,000 and 10,000 jobs instances, neither the initial model nor its relaxation succeeded in generating feasible solutions within the given time.

As noted earlier, load imbalance cannot be incorporated directly into the CP model without significantly reducing its scalability. Nevertheless, the corresponding metric is still reported, using the workload distribution observed in the CP solutions, to illustrate that the two objectives of the problem are not necessarily aligned. In practice, a schedule that performs well in terms of earliness and tardiness may still exhibit substantial disparities in the allocated workload.

Baselines from the metaheuristic. The second set of experiments evaluates the standalone constructive metaheuristic, which generates complete DHFFS schedules without invoking any CP models. The metaheuristic is applied to the same set of evaluation instances. In contrast to the CP model, it simultaneously considers the minimisation of both total earliness/tardiness and load imbalance.

Table 1 summarises the results of both sets of experiments. For each method, we report the total earliness and tardiness (in $\times 10^6$ minutes), shown in the column 'total E/T ($\times 10^6$)', as well as the Normalised Load Imbalance index $\mathrm{LI}^{\mathrm{norm}} = \frac{(\max_f L_f - \min_f L_f)}{\sum_f L_f}$, where L_f denotes the total scheduled processing load assigned to provider f. A value of $\mathrm{LI}^{\mathrm{norm}} = 0$ corresponds to a perfectly balanced allocation (all providers carry the same load), whereas values close to 1 indicate extreme imbalance, with almost all work concentrated in a single provider.

For the CP model, which was optimised only for the first objective, we report the best objective value found after 1,200, 2,400, and 3,600 seconds, as shown in the corresponding columns. Since the metaheuristic completed within a few minutes for all datasets, we report its final objective values directly, along with the total elapsed time in the 'Time' column (in seconds).

Results of hybrid RL-CP. Table 2 reports the detailed performance of the hybrid RL-CP policy for each evaluation instance. For small and medium instances (500, 800, 1,000 and 2,000 jobs) the method attains very low total earliness/tardiness, ranging from 0.003×10^6 to 0.010×10^6 minutes, with solve times between approximately 86 and 342 seconds. The normalised load-imbalance index remains below 0.025 in all these cases, indicating nearly perfectly balanced schedules across the two factories.

For the larger 5000- and 10000-job instances, the RL-CP policy continues to scale nicely. Total E/T lies between 0.226×10^6 and 0.536×10^6 minutes, while solve times range from approximately 860 to 1,780 seconds (well under 30 minutes). The corresponding normalised load-imbalance values remain modest, in the interval $[0.049, 0.058]$. Overall, the learned policy maintains low E/T and fair distribution, even as the instance size increases by an order of magnitude, demonstrating robust scalability of our RL-CP approach.

Comparing the results of Tables 1 and 2, for instances with 500-2,000 jobs, the best CP solutions (over all time checkpoints) obtain total E/T values between 0.09×10^6 and 3.5×10^6 minutes, while the metaheuristic baseline lies in a similar range (1-3 $\times 10^6$ minutes). In contrast, the hybrid RL-CP policy in achieves E/T values between 0.003×10^6 and 0.010×10^6 minutes on the corresponding instances, i.e., one to two orders of magnitude smaller than both CP and the metaheuristic. For 5000- and 10000-job instances, the CP model cannot be ap-

Table 1. Results of the baseline methods

Instance	CP model				Standalone metaheuristic		
	total E/T (minutes)($\times 10^6$)			LI^{norm}	total E/T ($\times 10^6$)	LI^{norm}	Time
	1,200	2,400	3,600				
500_1	0.362	0.273	0.087	0.14	1.154	0.08	206
500_2	0.361	0.284	0.090	0.02	1.126	0.19	204
500_3	0.952	0.318	0.255	0.25	1.070	0.09	205
800_1	-	1.559	0.267	0.30	1.716	0.08	223
800_2	0.858	0.223	0.222	0.08	1.662	0.25	220
800_3	1.513	0.465	0.465	0.96	1.482	0.11	225
1000_1	-	1.816	1.546	0.42	1.996	0.08	236
1000_2	1.896	1.317	0.420	0.10	1.938	0.24	232
1000_3	1.798	1.798	1.798	0.88	1.685	0.13	240
2000_1*	3.479	3.479	3.479	0.52	3.161	0.05	240
2000_2*	3.645	3.645	3.399	0.40	1.989	0.43	240
2000_3*	3.382	3.382	3.382	0.91	2.006	0.09	240
5000_1	-	-	-	-	1.343	0.02	240
5000_2	-	-	-	-	2.556	0.04	240
5000_3	-	-	-	-	2.246	0.05	240
10000_1	-	-	-	-	17.804	0.03	427
10000_2	-	-	-	-	14.960	0.00	240
10000_3	-	-	-	-	29.972	0.00	387

*For instances of 2,000 jobs, a relaxation of the CP model has been solved.

Table 2. Hybrid RL-CP performance per evaluation instance

Instance	Objective values		Time
	total E/T (minutes)($\times 10^6$)	LI^{norm}	
500_1	0.003	0.00	86
500_2	0.003	0.00	87
500_3	0.003	0.00	86
800_1	0.004	0.01	139
800_2	0.004	0.01	139
800_3	0.004	0.01	136
1000_1	1.038	0.02	170
1000_2	0.005	0.01	173
1000_3	0.005	0.01	170
2000_1	0.010	0.02	341
2000_2	0.010	0.02	340
2000_3	0.010	0.02	342
5000_1	0.304	0.05	865
5000_2	0.304	0.05	863
5000_3	0.226	0.06	870
10000_1	0.536	0.05	1777
10000_2	0.536	0.05	1781
10000_3	0.536	0.05	1781

plied to the original problem, whereas RL-CP still delivers high-quality schedules with E/T in the range 0.226-0.536 $\times 10^6$, compared to 1.343-29.972 $\times 10^6$ for the metaheuristic.

The hybrid approach also markedly improves fairness. The solutions of the CP model show that the load-imbalance objective is not always aligned with the minimisation of earliness/tardiness, as seen by the large deviations between the reported values, and the metaheuristic baseline exhibits normalised load-imbalance values between 0.02 and 0.43. By contrast, RL-CP attains $\mathrm{LI}^{\mathrm{norm}} = 0$ on all 500-job instances and stays below 0.025 for most 800, 1,000 and 2,000 job cases. Only for the largest 10,000-job instances we observe a trade-off: the metaheuristic achieves almost perfectly balanced solutions with very high E/T, whereas RL-CP accepts a modest imbalance ($\mathrm{LI}^{\mathrm{norm}} \approx 0.05$) in order to reduce E/T by more than an order of magnitude.

Regarding computation times, the metaheuristic remains the fastest method (about 200-430 seconds per instance), while RL-CP requires between 85 seconds on small instances and about 30 minutes on the largest 10,000 job cases. Given the substantial gains in both E/T and load imbalance, this extra CPU time is a reasonable cost in large-scale industrial settings. Overall, the experiments demonstrate that the proposed hybrid RL-CP policy not only exploits CP to scale to large instances, but also outperforms both the CP model and the metaheuristic baselines in most instances, within practical computation times.

5 Concluding remarks

This work presents an effective framework for integrating RL approaches, hybridised with standard optimisation models, for large-scale scheduling problems. As manufacturing demand in a MaaS environment grows to levels that cannot efficiently be handled by CP models or metaheuristics, delegating critical decomposition decisions to an RL agent offers faster and higher-quality solutions for both of our objectives.

Future work of this study strengthen this hybrid scheme along several directions. First, a more explicit fairness-aware training regime (e.g., stronger regularisation on the normalised load-imbalance, or hard balance constraints enforced by CP) could reduce the remaining imbalance for the largest instances without dramatically worsening E/T. Second, richer training curricula and longer training runs may further improve the learned policy, as the current model was trained for a limited number of epochs and is unlikely to have fully saturated its performance. Finally, learning to adapt CP search parameters dynamically during training could combine the strong optimality guarantees of CP with the scalability and flexibility of RL, leading to even better solutions on industrial-scale scheduling problems.

Acknowledgment. This research has been motivated and financially supported by Horizon Europe, as part of the Tec4MaaSEs research and innovation action, Grant Agreement No. 101138517.

References

1. Armstrong, E., Garraffa, M., O'Sullivan, B., Simonis, H.: The hybrid flexible flow-shop with transportation times. 27th International Conference on Principles and Practice of Constraint Programming (CP 2021) **210**, 1–18 (2021).
2. Cappart, Q., Moisan, T., Rousseau, L-M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. Proceedings of the AAAI conference on artificial intelligence **35**(5), 3677–3687 (2021).
3. Chalumeau, F., Coulon, I., Cappart, Q., Rousseau, L-M.: SeaPearl: A Constraint Programming Solver Guided by Reinforcement Learning. International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2021), 392–409 (2021).
4. Gholami, H., Sun, H.: Toward automated algorithm configuration for distributed hybrid flow shop scheduling with multiprocessor tasks. Knowledge-Based Systems **264**, 110309 (2023).
5. IBM ILOG CPLEX Optimization Studio Documentation: State functions, https://www.ibm.com/docs/en/icos/22.1.2?topic=scheduling-state-functions. Last updated: 2025-08-19.
6. Jiang, Y., Cao, Z., Zhang, J.: Learning to Solve 3-D Bin Packing Problem via Deep Reinforcement Learning and Constraint Programming. IEEE transactions on cybernetics **53**(5), 2864–2875 (2023).
7. Lee, J., Kao, H-A., Yang, S.: Service Innovation and Smart Analytics for Industry 4.0 and Big Data Environment. Procedia CIRP **16**, 3–8 (2014).
8. Li, R., Gong, W., Wang, L., Lu, C., Pan, Z., Zhang, X.: Double DQN-based coevolution for green distributed heterogeneous hybrid flowshop scheduling with multiple priorities of jobs. IEEE Transactions on Automation Science and Engineering **21**(4), 6550–6562 (2023).
9. Li, Y., Li, X., Gao, L., Zhang, B., Pan, Q-K., Tageriten, M.F., Meng, L.: A discrete artificial bee colony algorithm for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. International Journal of Production Research **59**(13), 3880–3899 (2021).
10. Li, Y., Yu, C.: Flexible Job Shop Scheduling with Job Precedence Constraints: A Deep Reinforcement Learning Approach. Journal of Manufacturing and Materials Processing **9**(7), 216 (2025).
11. Meng, L., Gao, K., Ren, Y., Zhang, B., Sang, H., Chaoyong, Z.: Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. Swarm and Evolutionary Computation **71**, 101058 (2022).
12. Naderi, B., Gohari, S., Yazdani, M.: Hybrid flexible flowshop problems: Models and solution methods. Applied Mathematical Modelling **38**(24), 5767–5780 (2014).
13. Pan, Q-K., Gao, L., Li, X-Y., Gao, K-Z.: Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. Applied Mathematics and Computation **303**, 89–112 (2017).
14. Pan, Q.-K., Ruiz, R., Alfaro-Fernández, P.: Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. Computers & Operations Research **80**, 50—60 (2017).
15. Pugliese, V., Ferreira, O., Faria, F.: Hybrid Flow Shop Scheduling through Reinforcement Learning: A systematic literature review. Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing, 1240–1249 (2025).

16. Ruiz, R., Vázquez-Rodríguez, J.A.: The hybrid flow shop scheduling problem. European Journal of Operational Research **205**(1), 1–18 (2010).
17. Shao, W., Shao, Z., Pi, D.: Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. Knowledge-Based Systems **194**, 105527 (2020).
18. Smit, I.G., Wu, Y., Troubil, P., Zhang, Y., Nuijten, W.P.M.: Neural Combinatorial Optimization for Stochastic Flexible Job Shop Scheduling Problems. Proceedings of the 39th AAAI Conference on Artificial Intelligence **39**(25), 26678–26687 (2025).
19. Sun, X., Shen, W., Fan, J., Vogel-Heuser, B., Zhang, C.: An improved non-dominated sorting genetic algorithm II for distributed heterogeneous hybrid flow-shop scheduling with blocking constraints. Journal of Manufacturing Systems **77**, 990–1008 (2024).
20. Tassel, P., Gebser, M., Schekotihin, K.: An end-to-end reinforcement learning approach for job-shop scheduling problems based on constraint programming. Proceedings of the International Conference on Automated Planning and Scheduling **33**(1), 614–622 (2023).
21. Xi, B., Lei, D.: Q-learning-based teaching-learning optimization for distributed two-stage hybrid flow shop scheduling with fuzzy processing time. Complex System Modeling and Simulation **2**(2), 113-129 (2022).
22. Ying, K-C., Lin, S-W.: Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks. Expert Systems with Applications **92**, 132–141 (2018).
23. Zhang, W., Bao, X., Geng, H., Zhang, G., Gen, M.: Graph neural network and expert-guided deep reinforcement learning for solving flexible job-shop scheduling problem. Computers & Operations Research **183**, 107155 (2025).
24. Zhang, W., Li, C., Gen, M., Yang, W., Zhang, G.: A multiobjective memetic algorithm with particle swarm optimization and Q-learning-based local search for energy-efficient distributed heterogeneous hybrid flow-shop scheduling problem. Expert Systems with Applications **237**(C), 121570 (2024).
25. Zhu, Q., Gao, K., Huang, W., Ma, Z., Slowik, A.: Q-Learning-Assisted Meta-Heuristics for Scheduling Distributed Hybrid Flow Shop Problems. Computers, Materials & Continua **80**(3), 3573 (2024).