Unrelated Parallel Machine Scheduling under Non-Resumable Maintenance: A Surrogate-Assisted Genetic Algorithm

Fatima Iken[1], Abdennour Azerine[1], Mahmoud Golabi[1], Lhassane Idoumghar[1]
Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France.

***Abstract*— We study the problem of scheduling independent jobs on unrelated parallel machines subject to non-resumable maintenance constraints, with the goal of minimizing the makespan. The presence of planned maintenance windows restricts machine availability and tightly couples assignment and sequencing decisions, making solution evaluation particularly costly. We address this challenge through a hybrid genetic algorithm built on a decomposition principle: the genetic layer searches the space of job-to-machine assignments, while each induced single-machine subproblem is solved exactly via a dynamic programming decoder formulated as a 0–1 knapsack problem. Although this decoder provides rigorous fitness guarantees, it constitutes the dominant computational bottleneck within the search.**

To overcome this cost, a Random Forest surrogate is embedded into the genetic algorithm to predict makespan values and selectively bypass exact evaluations. A central contribution of this work is the investigation of surrogate input representations: we compare a direct assignment-based encoding, in which the job-to-machine assignment vector is fed directly to the model, against a feature-based encoding that summarizes each machine's workload through aggregated per-machine statistics, comprising processing time statistics, a greedy makespan estimate, and the load accumulated prior to the maintenance window. Experiments on a set of generated instances demonstrate that the surrogate-assisted algorithm substantially reduces computation time while preserving solution quality, and that the choice of input representation has a measurable impact on prediction accuracy and overall search performance.

I. INTRODUCTION

Scheduling involves the allocation of jobs to limited processing resources over time, subject to operational constraints and one or more performance objectives [1]. Among the most relevant problem classes in manufacturing and production planning is parallel machine scheduling, where multiple machines operate simultaneously, and jobs must be distributed efficiently across them [2]. Parallel machine environments are commonly classified as identical, uniform, or unrelated, with the unrelated case representing the most general setting since processing times are specific to each job-machine pair [3]. Across all these variants, makespan minimization stands as one of the most widely studied objectives in scheduling, as it directly reflects system throughput and overall resource utilization [4].

Classical scheduling theory typically assumes that machines are continuously available [5], yet this assumption rarely holds in practice since industrial systems require periodic preventive maintenance to sustain operational reliability. When maintenance is non-resumable, preemption is forbidden, and every job must execute without interruption within a single window of machine availability [6]. This constraint breaks the order-independence property commonly exploited in makespan optimization and embeds a combinatorial knapsack structure into the assignment process [1].

The unrelated parallel-machine makespan problem $R||C_{\max}$ is known to be NP-hard, and Lenstra et al. [7] established that no polynomial-time algorithm can guarantee an approximation ratio below $3/2$ unless $P = NP$. Surveys by Schmidt [8] and Lee [9] provide systematic classifications of availability constraints and contrast resumable with non-resumable interruptions. For the non-resumable single-machine case, Lee [10] showed equivalence to the subset sum problem, establishing weak NP-hardness.

The broader literature on maintenance-constrained scheduling has concentrated primarily on single-machine settings [11], relying on heuristics [12], mixed-integer linear programming [13], and dynamic programming [14]. Extensions to parallel machines have largely targeted identical or uniform configurations [15], leaving unrelated-machine settings comparatively understudied. Kurt and Çetinkaya [16] tackled unrelated parallel machines with a single non-resumable maintenance period using a mixed-integer model paired with a constructive heuristic, though exact methods remain difficult to scale to larger instances [17].

Maintenance constraints in parallel-machine environments give rise to two coupled decision layers: a global job-to-machine assignment and a per-machine sequencing problem around maintenance windows. For fixed assignments, each machine subproblem reduces to a capacity-constrained job selection task solvable by dynamic programming [18], yet the state space expands exponentially as the number of machines grows [19], [20]. This scalability limitation motivates hybrid methods that pair a global metaheuristic with exact dynamic programming at the machine level. Glass et al. [21] established early evidence that such hybrid designs effectively balance solution quality and runtime, a finding confirmed by more recent work addressing complex operational constraints [22]. Repeated exact evaluations within these frameworks, however, impose a heavy computational burden, and surrogate-assisted optimization alleviates this cost by substituting machine learning predictions for expensive fitness computations [23], [24]. Despite their promise, surrogate-assisted evolutionary methods for unrelated parallel machines with non-resumable maintenance have received little attention in the literature.

Motivated by this gap, we study the unrelated parallel-machine scheduling problem with one non-resumable maintenance window per machine under the makespan criterion. Given the NP-hardness of job-to-machine assignment [25], we propose a hybrid genetic algorithm that drives evolutionary search over the assignment space while resolving each induced machine-level subproblem exactly through dynamic programming. To reduce the computational overhead of repeated exact evaluations, a Random Forest surrogate model is embedded into the algorithm, learning online from previously evaluated solutions and predicting makespan for candidate assignments, with selective exact retraining to sustain predictive accuracy [23]. A central contribution of this work is the systematic comparison of two surrogate input representations: a direct assignment-based encoding and a feature-based encoding that condenses each machine's workload into aggregated statistics. Together, these components address the methodological gap in maintenance-constrained unrelated parallel machine scheduling.

The remainder of this paper is organized as follows. Section II formulates the problem. Section III presents the proposed hybrid genetic algorithm. Section IV details the single-machine evaluation via dynamic programming. Section V describes the surrogate-assisted evaluation framework and the two input representations. Section VII reports the computational experiments and discusses the results. Section VIII concludes the paper.

II. Problem Definition

We consider a scheduling problem involving a set of n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and a set of m unrelated parallel machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j requires a machine-dependent processing time p_{ij} when executed on machine M_i, for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. All jobs become available simultaneously at time zero, each must be assigned to exactly one machine, and no preemption is permitted.

Each machine M_i is subject to a planned maintenance window $[\tau_{1i}, \tau_{2i}]$, during which it remains unavailable for processing, where τ_{1i} and τ_{2i} denote the start and end of the maintenance period, respectively. Jobs are non-resumable: if a job starts on machine M_i but cannot complete before τ_{1i}, it cannot be interrupted and must be entirely rescheduled to begin no earlier than τ_{2i}.

The objective is to construct a feasible schedule that minimizes the makespan $C_{\max}$. In the standard three-field scheduling notation [26], this problem is denoted $Rm \mid nr\text{-}maint \mid C_{\max}$.

III. Proposed Approach

We employ a genetic algorithm (GA) as the core search strategy [27]. The evolutionary procedure exclusively controls the job-to-machine assignment σ, while all aspects related to the non-resumable maintenance constraint are resolved within the fitness evaluation. For any fixed assignment σ, the evaluation determines the minimum completion time for each machine given its maintenance window, computing the global makespan as the maximum across all machines.

A. Solution encoding

Each solution (chromosome) is encoded as a vector $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, where each gene $\sigma_j \in \{1, \ldots, m\}$ specifies the machine assigned to job J_j. This direct integer encoding, tailored to the $R|nr\text{-}maint|C_{\max}$ problem, guarantees feasibility. Every job is assigned to exactly one machine, and each machine processes its assigned jobs whenever it is not under maintenance.

For each machine M_i and chromosome σ, the assigned job set is $J_i(\sigma) = \{J_j \in \mathcal{J} \mid \sigma_j = i\}$, processed in a fixed order. Jobs overlapping the maintenance start are delayed until τ_{2i}, ensuring adherence to the non-resumable constraint without requiring repair mechanisms during evolutionary search. This isolates assignment from machine-level scheduling, enabling independent fitness evaluation per machine.

B. Genetic Algorithm Framework

The GA employs a generational scheme with elitism. Following extensive preliminary parameter tuning, it initializes a diverse population of 70 candidates by uniformly sampling genes $\sigma_j \in \{1, \ldots, m\}$.

Fitness is evaluated by decomposing the problem into m independent subproblems, computing completion times $C^i_{\max}(\sigma)$ under intervals $[\tau_{1i}, \tau_{2i}]$ to find the global makespan:

$$C_{\max}(\sigma) = \max_i C^i_{\max}(\sigma) \tag{1}$$

Search is driven by tournament selection, uniform crossover ($p_c = 0.75$), and swap mutation ($p_m = 0.10$). Elitism preserves the best solution, which also serves as a reference for surrogate-assisted evaluation. The search terminates after 1000 generations or 25 consecutive generations without improvement.

IV. Machine-Level Problem Formulation

This section evaluates a fixed assignment σ by formulating the single-machine subproblem, establishing its reduction to a knapsack problem resulting in weak NP-hardness, and detailing the exact dynamic programming computation.

A. Subproblem Structure for Fixed Assignments

For machine M_i, let $J^i(\sigma)$ be the assigned jobs to machine i, $k_i = |J^i(\sigma)|$, and $P_i = \{\sum_j p_{ij} : J_j \in J^i(\sigma)\}$ be the processing load on machine i. Since all jobs are available at time zero without setup times, scheduling reduces to selecting which jobs complete before the maintenance starts τ_{1i}.

If total load $P_i \leq \tau_{1i}$, all jobs finish before maintenance, yielding $C^i_{\max}(\sigma) = P_i$. If the load does not fit, a subset $B_i \subseteq J^i(\sigma)$ completes before maintenance, and the remaining jobs start after τ_{2i}, resulting in:

$$C^i_{\max}(\sigma) = \tau_{2i} + \sum_{J_j \in J^i(\sigma) \setminus B_i} p_{ij} \tag{2}$$

B. Optimal Subset Selection through Knapsack Reduction

Proposition 1. *Given fixed σ and machine M_i, minimizing $C^i_{\max}(\sigma)$ is equivalent to maximizing the processing time executed before maintenance, subject to a capacity constraint:*

$$\max_{B_i \subseteq \mathcal{J}^i(\sigma)} \sum_{J_j \in B_i} p_{ij} \quad \textit{s.t.} \quad \sum_{J_j \in B_i} p_{ij} \leq \tau_{1i} \tag{3}$$

Proof: If the load exceeds τ_{1i}, let B_i be jobs completed before τ_{1i}. Because jobs are non-preemptive, available at time zero, and lack setup times, jobs in B_i can be sequentially processed from time zero, and remaining jobs back-to-back from τ_{2i}, in any order. The completion time is $\tau_{2i} + \sum_{J_j \in J^i(\sigma) \setminus B_i} p_{ij}$. Minimizing this is equivalent to maximizing pre-maintenance processing time under the constraint $\sum_{J_j \in B_i} p_{ij} \leq \tau_{1i}$.

Corollary 2. *The single-machine subproblem induced by a fixed assignment with one non-resumable maintenance interval is weakly NP-hard [28].*

Proof: Proposition 1 implies the subproblem equates to a 0–1 knapsack problem [29].

C. Optimal Evaluation via Dynamic Programming

The exact optimum is computed using a classical pseudo-polynomial dynamic programming (DP) method with $\mathcal{O}(k_i \tau_{1i})$ time and memory complexity [30]. The DP returns the best packed value and, via backtracking, the corresponding subset.

For large τ_{1i}, exact evaluation becomes computationally expensive, motivating surrogate-assisted strategies.

Exact Subproblem Evaluation: Let $L_i(\sigma) = \sum_{J_j \in J_i(\sigma)} p_{ij}$ be the total load on machine M_i, and $V_i(\sigma)$ denote the optimal knapsack value. Consequently:

$$C^i_{\max}(\sigma) = \begin{cases} L_i(\sigma), & \text{if } L_i(\sigma) \leq \tau_{1i}, \\ \tau_{2i} + L_i(\sigma) - V_i(\sigma), & \text{otherwise}, \end{cases} \tag{4}$$

and the overall makespan is given by:

$$C_{\max}(\sigma) = \max_i C^i_{\max}(\sigma) \tag{5}$$

V. Surrogate-Based Fitness Approximation

Repeated dynamic programming evaluations represent the dominant computational cost within the genetic algorithm. To address this, a surrogate-assisted fitness evaluation mechanism is integrated to selectively replace exact computations with fast predictions, while maintaining reliable search performance through controlled safeguards. The following subsections detail the training strategy, the evaluation policy, and the surrogate input representations investigated in this work.

A. Surrogate Model and Training Strategy

A Random Forest surrogate [31], is employed to approximate the mapping $f : \sigma \mapsto C_{\max}(\sigma)$, where $\sigma = (\sigma_1, \ldots, \sigma_n) \in \{1, \ldots, m\}^n$ denotes a job-to-machine assignment vector and $C_{\max}(\sigma) \in \mathbb{R}_+$ is the exact makespan obtained through the evaluation procedure described in Section IV. The surrogate is trained online during the genetic algorithm using pairs $(\phi(\sigma), y)$, where $\phi(\sigma)$ denotes the chosen input encoding of the assignment — either ϕ_{assign} or ϕ_{feat} as defined in Section V-B — and $y = C_{\max}(\sigma)$ is the corresponding exact makespan. To control training cost and mitigate concept drift as the search progresses, the training dataset is maintained as a sliding window containing at most $N_{\max} = 2000$ of the most recent exact evaluations, with older samples discarded. This online data management strategy preserves model adaptability to the current search region while maintaining bounded computational overhead.

B. Surrogate Input Representation

A key design decision in the surrogate framework concerns the choice of input representation. Two encoding strategies are evaluated and compared experimentally in Section VI.

Assignment-based encoding: uses the raw integer assignment vector $\sigma = (\sigma_1, \ldots, \sigma_n) \in \{1, \ldots, m\}^n$ directly as the surrogate input, where each entry σ_j indicates the machine assigned to job J_j. This representation has dimension n and maintains full consistency with the genetic encoding.

Feature-based encoding: aggregates per-machine statistics derived from the assignment, yielding a compact representation that abstracts away job-level details. For each machine M_i, six features are extracted from the set of jobs assigned to it: the minimum, maximum, variance, and mean of the corresponding processing times $\{p_{ij} : \sigma_j = i\}$, the machine makespan estimated by a greedy heuristic $\tilde{C}_i^{\text{greedy}}$ obtained by sequencing the assigned jobs in non-increasing order of processing time, and the total processing load scheduled before the maintenance start τ_{1i}. The resulting feature vector has dimension $m \times 6$.

Formally, the two input mappings are:

$$\phi_{\text{assign}}(\sigma) = \sigma \in \{1, \ldots, m\}^n, \tag{6}$$

$$\phi_{\text{feat}}(\sigma) = \left[\mu_i,\ \min_i,\ \max_i,\ \text{var}_i,\ \tilde{C}_i^{\text{greedy}},\ L_i^{\tau}\right]_{i=1}^{m} \in \mathbb{R}^{m \times 6}, \tag{7}$$

where μ_i, $\min_i$, $\max_i$, and var_i denote the mean, minimum, maximum, and variance of processing times on machine M_i, respectively, and

$$L_i^{\tau} = \sum_{j:\, \sigma_j = i} p_{ij} \cdot \mathbf{1}[p_{ij} \leq \tau_{1i}] \tag{8}$$

is the total processing load feasibly schedulable before the maintenance window on machine M_i. The two representations are assessed using the Random Forest surrogate to determine which encoding best supports accurate makespan prediction within the genetic algorithm.

C. Evaluation Policy and Safeguards

The surrogate model becomes operational after a three-generation initialization period, during which every candidate solution is evaluated using exact evaluation to build the foundational training dataset, which we denote by E_0. After activation, the model generates predicted makespan values

$\tilde{C}_{\max}(\sigma)$ for the majority of candidate assignments and is periodically retrained at ten-generation intervals to reflect the evolving search trajectory.

Exact evaluation is triggered through a dual-criterion mechanism: first, any solution whose surrogate estimate falls below the current best-known makespan $C_{\max}^{\text{best}}$ receives mandatory exact evaluation to guarantee reliable identification of superior solutions; second, a probability $\rho = 0.1$ independently enforces exact evaluation to preserve exploratory behavior and mitigate potential systematic prediction errors.

Each exact evaluation immediately contributes the pair $(\sigma, C_{\max}(\sigma))$ to the training repository, with concurrent updates to the best-known solution when improvements occur. For all other candidates, the surrogate prediction serves directly as the fitness metric. This framework (Algorithm 1) maintains efficiency while ensuring that promising solutions are rigorously verified.

Algorithm 1 Surrogate-assisted fitness evaluation

Require: Candidate assignment σ, current generation g, best-so-far makespan $C_{\max}^{\text{best}}$, Random Forest surrogate $\mathcal{F}$
Ensure: Fitness value $f(\sigma)$ and updated training set
if $g < E_0$ **or** $\mathcal{F}$ not initialized **then**
  $f \leftarrow \textsc{ExactEval}(\sigma)$
  Add (σ, f) to training set
  $C_{\max}^{\text{best}} \leftarrow \min\{C_{\max}^{\text{best}}, f\}$
  return f
end if
if $(g - E_0) \bmod R = 0$ **and** first evaluation in generation g **then**
  Retrain $\mathcal{F}$ on current training set
end if
$\tilde{f} \leftarrow \mathcal{F}.\textsc{Predict}(\sigma)$
$u \sim \text{Uniform}(0, 1)$
if $\tilde{f} < C_{\max}^{\text{best}}$ **or** $u < \rho$ **then**
  $f \leftarrow \textsc{ExactEval}(\sigma)$
  Add (σ, f) to training set
  $C_{\max}^{\text{best}} \leftarrow \min\{C_{\max}^{\text{best}}, f\}$
  return f
else
  return $\tilde{f}$
end if

VI. Computational Experiments

This section presents the computational experiments conducted to evaluate the proposed approach. We assess the genetic algorithm under both exact and surrogate-assisted fitness evaluation, and compare the two input representations of the Random Forest surrogate across a set of synthetically generated instances. The experiments are organized as follows: Section VI-A describes the instance generation procedure and maintenance setup, Section VI-B details the experimental configuration, and the results are analyzed and discussed in Section VII.

A. Instance Generation and Maintenance Setup

Each instance is characterized by a pair (m, n) denoting the number of machines and jobs, respectively. Four job sizes and four machine counts are considered, yielding sixteen configurations: $n \in \{20, 50, 100, 200\}$ and $m \in \{2, 3, 4, 5\}$. For each configuration, processing times p_{ij} are drawn independently for each machine-job pair from a prescribed interval corresponding to three scale levels: small $p_{ij} \sim \mathcal{U}(1, 100)$, medium $p_{ij} \sim \mathcal{U}(20, 200)$, and large $p_{ij} \sim \mathcal{U}(100, 1000)$.

Maintenance windows are designed to induce heterogeneous and non-trivial knapsack subproblems at the machine level. For each machine M_i, the total hypothetical load $L_i = \sum_{j=1}^{n} p_{ij}$ is first computed. The maintenance start time is set to $\tau_{1i} = \lfloor \alpha L_i \rfloor$, where $\alpha \sim \mathcal{U}(0.3, 0.7)$, placing the maintenance window within the central portion of the processing horizon and avoiding degenerate boundary placements. The maintenance duration is drawn uniformly from $[d_{\min}, d_{\max}]$, with $d_{\min} = \max\{p_{\min}, \lfloor 0.15\, L_i \rfloor\}$ and $d_{\max} = \max\{d_{\min}+1, \lfloor 0.40\, L_i \rfloor\}$, and the maintenance end time is accordingly $\tau_{2i} = \tau_{1i} + \text{duration}$. Tying all maintenance parameters to L_i ensures that the relative disruption remains consistent across instance sizes, while the machine-dependent variation of L_i naturally produces heterogeneous maintenance windows in the unrelated-machine setting.

B. Experimental Setup

All experiments were carried out on a workstation equipped with an Intel(R) Xeon(R) @ 2.71 GHz, 128 GB RAM, running Windows 11 Pro for Workstations and Python 3.14. The proposed genetic algorithm is evaluated under two fitness evaluation modes: an exact mode relying entirely on dynamic programming, and a surrogate-assisted mode in which a Random Forest model selectively replaces exact evaluations as described in Section V. Within the surrogate-assisted mode, the two input representations are compared across all generated instances to assess their respective impact on solution quality and computational efficiency.

VII. Results and Discussion

This section reports and discusses the results obtained from the computational experiments described in Section VI-B. We examine the surrogate-assisted genetic algorithm by contrasting the feature-based and assignment-based input representations across all generated instances in terms of solution quality and prediction accuracy.

a) Comparison of Input Representations: Tables I and II compare the surrogate-assisted genetic algorithm under the feature-based encoding with the assignment-based GA+DP baseline. The cost of dynamic programming grows rapidly with instance size, reaching 10,269.7 seconds at (n=200, m=2) for large instances, which limits its applicability to larger problems. The feature-based encoding reduces this cost substantially: at ($n = 100,\ m = 2$), runtime decreases from $3,247.8$ to 766.8 seconds. This efficiency is attributable to the nine-dimensional feature representation, which encodes the job-machine interactions governing the makespan and thereby permits accurate surrogate prediction without exhaustive evaluation.

TABLE I: Baseline GA+DP exact evaluation: mean makespan and execution time per instance configuration across processing time scales .

n	m	Small		Medium		Large	
		$C_{\max}$	Time (s)	$C_{\max}$	Time (s)	$C_{\max}$	Time (s)
20	2	481.8	174.1	1115.1	171.0	5532.0	184.3
20	3	223.3	428.1	651.8	361.9	3284.0	377.5
20	4	160.2	380.5	468.6	366.5	2247.6	374.1
20	5	142.7	395.7	389.8	401.0	1976.7	413.4
50	2	1015.6	770.0	2803.1	601.5	13758.8	1099.3
50	3	629.9	695.8	1713.2	593.8	8579.6	810.5
50	4	461.4	569.5	1328.2	527.4	6476.4	661.2
50	5	375.6	536.3	1064.8	443.2	5527.6	550.4
100	2	2358.3	1114.7	5565.8	1087.6	28970.3	3247.8
100	3	1377.5	960.6	3597.1	807.2	18761.5	1843.2
100	4	1028.0	840.5	2772.5	647.9	14366.8	1433.7
100	5	856.0	692.4	2330.2	610.6	11375.7	1165.4
200	2	5444.2	2595.3	12148.2	2803.5	60911.1	10269.7
200	3	2959.2	2279.7	7710.2	1763.5	39582.3	6555.1
200	4	2207.6	1908.3	5928.9	1363.4	29221.1	4744.1
200	5	1824.5	1781.1	4830.1	1002.0	24038.3	3656.6

b) Evaluation Metrics: Accuracy and MAE: The prediction-quality metrics in Table II are consistent with this advantage. The feature-based encoding φ_{feat} often attains accuracy above 0.9, particularly at low machine counts, whereas the assignment-based encoding φ_{assign} is both lower and more variable, decreasing to 0.20 on the smallest instances. This gap narrows as instance size increases but does not close. A corresponding reduction in MAE is observed for φ_{feat} at low machine counts (m=2, 3), where the aggregated machine-level statistics provide a compact representation that the Random Forest can fit accurately. As the number of machines grows (m=4, 5) and instances become larger, the MAE advantage erodes and reverses: on high-m configurations—most consistently in the large class— φ_{assign} attains the lower MAE, suggesting that at this scale the raw assignment vector already exhibits sufficient regularity for the surrogate. Overall, the structured feature representation yields more accurate makespan predictions and, in turn, more reliable search guidance than the direct assignment encoding across most of the configuration space.

VIII. Conclusion

This work investigated a hybrid genetic algorithm for the unrelated parallel-machine scheduling problem with non-resumable maintenance constraints, combining evolutionary search over the job-to-machine assignment space with exact dynamic programming decoding at the machine level. Although exact evaluation guarantees rigorous fitness calculation, its repeated application across large populations introduces a significant computational bottleneck. A Random Forest surrogate model is therefore embedded into the genetic algorithm to selectively substitute costly exact evaluations with fast predictions, reducing this overhead without compromising the overall search quality.

TABLE II: Mean Makespan, Execution Time, Accuracy, and MAE: Feature-Based and Assignment-Based Encoding.

n	m	Makespan		Time (s)		Accuracy		MAE	
		Feat.	Asgn.	Feat.	Asgn.	Feat.	Asgn.	Feat.	Asgn.
Small instances									
20	2	481.8	481.8	138.9	174.1	0.934	0.420	22.85	73.07
20	3	226.8	223.3	140.4	428.1	0.862	0.212	31.13	54.41
20	4	167.3	160.2	141.6	380.5	0.786	0.224	32.71	42.31
20	5	143.3	142.7	142.7	395.7	0.655	0.199	33.01	36.45
50	2	1012.1	1015.6	163.0	770.0	0.960	0.320	47.14	113.44
50	3	622.4	629.9	164.1	695.8	0.891	0.437	49.10	61.07
50	4	432.3	461.4	165.2	569.5	0.782	0.422	47.50	49.94
50	5	339.6	375.6	164.9	536.3	0.710	0.475	43.66	47.89
100	2	2322.2	2358.3	254.0	1114.7	0.970	0.660	70.58	149.44
100	3	1317.1	1377.5	254.9	960.6	0.925	0.684	67.62	101.74
100	4	955.4	1028.0	252.7	840.5	0.819	0.672	65.53	58.98
100	5	759.1	856.0	253.3	692.4	0.758	0.695	59.09	54.57
200	2	5229.9	5444.2	634.5	2595.3	0.970	0.789	102.32	235.06
200	3	2895.6	2959.2	634.5	2279.7	0.921	0.765	94.38	129.84
200	4	2101.2	2207.6	623.9	1908.3	0.824	0.819	83.34	66.11
200	5	1722.5	1824.5	636.7	1781.1	0.832	0.767	77.32	60.01
Medium instances									
20	2	1115.1	1115.1	139.3	171.0	0.898	0.515	35.55	72.69
20	3	652.7	651.8	138.8	361.9	0.804	0.302	47.07	70.01
20	4	460.7	468.6	138.8	366.5	0.706	0.257	52.71	70.50
20	5	387.2	389.8	139.4	401.0	0.656	0.227	50.36	65.19
50	2	2784.8	2803.1	162.5	601.5	0.916	0.379	65.75	101.30
50	3	1640.8	1713.2	155.2	593.8	0.854	0.480	84.93	102.21
50	4	1202.0	1328.2	150.8	527.4	0.790	0.475	87.49	95.80
50	5	956.6	1064.8	147.1	443.2	0.754	0.558	86.31	91.23
100	2	5478.9	5565.8	253.8	1087.6	0.937	0.739	100.27	129.41
100	3	3402.2	3597.1	210.5	807.2	0.890	0.776	123.90	126.64
100	4	2613.7	2772.5	190.6	647.9	0.855	0.737	112.92	116.26
100	5	2163.1	2330.2	182.3	610.6	0.820	0.631	113.70	105.96
200	2	11981.4	12148.2	611.2	2803.5	0.945	0.831	156.82	161.68
200	3	7463.6	7710.2	472.5	1763.5	0.907	0.846	165.59	143.40
200	4	5741.8	5928.9	376.2	1363.4	0.878	0.766	153.79	131.29
200	5	4628.1	4830.1	330.9	1002.0	0.865	0.821	140.69	121.14
Large instances									
20	2	5532.0	5532.0	155.1	184.3	0.880	0.505	182.23	347.76
20	3	3321.6	3284.0	149.5	377.5	0.792	0.322	241.15	353.12
20	4	2174.1	2247.6	146.9	374.1	0.726	0.283	259.28	350.86
20	5	1926.0	1976.7	145.4	413.4	0.632	0.256	262.63	329.83
50	2	13664.1	13758.8	275.9	1099.3	0.913	0.398	330.17	473.27
50	3	8258.8	8579.6	228.1	810.5	0.847	0.483	407.94	507.22
50	4	5984.6	6476.4	203.6	661.2	0.808	0.461	426.71	481.11
50	5	5277.5	5527.6	189.2	550.4	0.749	0.610	407.71	471.92
100	2	28458.9	28970.3	766.8	3247.8	0.930	0.741	527.85	635.47
100	3	17806.2	18761.5	535.4	1843.2	0.892	0.787	602.21	634.77
100	4	13406.2	14366.8	429.3	1433.7	0.856	0.702	575.08	561.96
100	5	10466.0	11375.7	374.4	1165.4	0.819	0.738	547.45	531.48
200	2	59936.1	60911.1	2610.8	10269.7	0.944	0.905	778.29	757.03
200	3	38345.1	39582.3	1882.8	6555.1	0.914	0.853	802.77	730.54
200	4	28195.2	29221.1	1384.5	4744.1	0.874	0.818	804.73	658.85
200	5	23159.0	24038.3	1137.6	3656.6	0.860	0.802	691.96	587.29

The computational experiments demonstrate that the surrogate-assisted variant achieves significant reductions in runtime across all tested instance configurations, while preserving solution quality relative to the exact baseline. This confirms that surrogate-assisted evaluation constitutes a practical and effective strategy for mitigating the computational cost of dynamic programming within evolutionary scheduling frameworks. A central contribution of this work is the comparison of two surrogate input representations: a direct assignment-based encoding, which feeds the raw job-to-machine assignment vector into the model, and a feature-based encoding, which summarizes each machine's workload through aggregated per-machine statistics. The results show that the feature-based encoding consistently achieves better predictive accuracy and solution quality. This indicates that representing the assignment through structured machine-level features gives the surrogate more informative inputs, resulting in more accurate predictions and better search decisions.

Taken together, these results show that combining exact decomposition with surrogate-assisted evaluation and a carefully chosen input representation leads to a practical and efficient solution for the unrelated parallel machine scheduling problem with non-resumable maintenance constraints.

Several directions can be pursued to extend the present work. On the surrogate side, benchmarking a broader set of machine learning regression models would help identify which surrogate best fits this problem class. The feature-based encoding could also be improved by adding more problem-specific features, such as job criticality with respect to the maintenance window or machine load imbalance. From a problem scope perspective, the framework could be extended to handle multiple maintenance periods per machine, sequence-dependent setup times, or time-dependent processing times, all of which appear in industrial settings. Finally, extending the framework to a multi-objective setting, where makespan is optimized simultaneously with criteria such as total weighted tardiness or energy consumption, would further broaden the applicability of the proposed approach.

REFERENCES

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. Cham, Switzerland: Springer, 2016. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-26580-3

[2] T. Cheng and C. Sin, "A state-of-the-art review of parallel-machine scheduling research," *European Journal of Operational Research*, vol. 47, no. 3, pp. 271–292, 1990.

[3] J. Kaabi and Y. Harrath, "A survey of parallel machine scheduling under availability constraints," *International Journal of Computer and Information Technology*, vol. 3, no. 2, pp. 238–245, 2014.

[4] M. Ziaee, "Job shop scheduling with makespan objective: A heuristic approach," *International Journal of Industrial Engineering Computations*, vol. 5, no. 2, p. 273, 2014.

[5] J. Y. Leung, *Handbook of scheduling: algorithms, models, and performance analysis*. Chapman and Hall/CRC, 2004.

[6] J. Zou and J. Yuan, "Single-machine scheduling with maintenance activities and rejection," *Discrete Optimization*, vol. 38, p. 100609, 2020.

[7] J. K. Lenstra, D. B. Shmoys, and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical programming*, vol. 46, no. 1, pp. 259–271, 1990.

[8] G. Schmidt, "Scheduling with limited machine availability," *European Journal of Operational Research*, vol. 121, no. 1, pp. 1–15, 2000.

[9] E. Sanlaville and G. Schmidt, "Machine scheduling with availability constraints," *Acta Informatica*, vol. 35, no. 9, pp. 795–811, 1998.

[10] C.-Y. Lee, "Machine scheduling with an availability constraint," *Journal of global optimization*, vol. 9, no. 3, pp. 395–416, 1996.

[11] C. Low, M. Ji, C.-J. Hsu, and C.-T. Su, "Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance," *Applied Mathematical Modelling*, vol. 34, no. 2, pp. 334–342, 2010.

[12] S.-H. Lee and I.-G. Lee, "Heuristic algorithm for the single-machine scheduling with periodic maintenance," *Journal of Korean Institute of Industrial Engineers*, vol. 34, no. 3, pp. 318–327, 2008.

[13] A. A. Qamhan, A. Ahmed, I. M. Al-Harkan, A. Badwelan, A. M. Al-Samhan, and L. Hidri, "An exact method and ant colony optimization for single machine scheduling problem with time window periodic maintenance," *IEEE Access*, vol. 8, pp. 44 836–44 845, 2020.

[14] C. Sadfi, B. Penz, and C. Rapine, "A dynamic programming algorithm for the single machine total completion time scheduling problem with availability constraints," in *Eighth international workshop on project management and scheduling*, 2002.

[15] D.-L. Yang, T. Cheng, S.-J. Yang, and C.-J. Hsu, "Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities," *Computers & Operations Research*, vol. 39, no. 7, pp. 1458–1464, 2012.

[16] A. Kurt and F. C. Çetinkaya, "Unrelated parallel machine scheduling under machine availability and eligibility constraints to minimize the makespan of non-resumable jobs," *International Journal of Industrial Engineering and Management*, vol. 15, no. 1, pp. 18–33, 2024.

[17] J. Pacheco, F. Ángel-Bello, and A. Álvarez, "A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times," *Journal of Scheduling*, vol. 16, no. 6, pp. 661–673, 2013.

[18] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[19] P. Shi and D. Landa-Silva, "Dynamic programming with approximation function for nurse scheduling," in *International Workshop on Machine Learning, Optimization, and Big Data*. Springer, 2016, pp. 269–280.

[20] C. Koulamas and G. J. Kyparisis, "A classification of dynamic programming formulations for offline deterministic single-machine scheduling problems," *European Journal of Operational Research*, vol. 305, no. 3, pp. 999–1017, 2023.

[21] C. A. Glass, C. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search," *Mathematical and Computer Modelling*, vol. 20, no. 2, pp. 41–52, 1994.

[22] J. Adan, "A hybrid genetic algorithm for parallel machine scheduling with setup times: A comparative study of metaheuristics on large problem instances," *Journal of Intelligent Manufacturing*, vol. 33, no. 7, pp. 2059–2073, 2022.

[23] T. J. Ikonen, K. Heljanko, and I. Harjunkoski, "Surrogate-based optimization of a periodic rescheduling algorithm," *AIChE Journal*, vol. 68, no. 6, p. e17656, 2022.

[24] S. Liu, H. Wang, W. Peng, and W. Yao, "Surrogate-assisted evolutionary algorithms for expensive combinatorial optimization: a survey," *Complex & Intelligent Systems*, vol. 10, no. 4, pp. 5933–5949, 2024.

[25] L. Fanjul-Peyro and R. Ruiz, "Iterated greedy local search methods for unrelated parallel machine scheduling," *European Journal of Operational Research*, vol. 207, no. 1, pp. 55–69, 2010.

[26] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[27] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[28] W. Luo, L. Chen, and G. Zhang, "Approximation algorithms for scheduling with a variable machine maintenance," in *International Conference on Algorithmic Applications in Management*. Springer, 2010, pp. 209–219.

[29] W. Luo and F. Liu, "On single-machine scheduling with workload-dependent maintenance duration," *Omega*, vol. 68, pp. 119–122, 2017.

[30] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin: Springer, 2004.

[31] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.