A Genetic Algorithm with Surrogate-Assisted Dynamic Programming for Maintenance Constrained Parallel Machine Scheduling

Fatima Iken[1 [0009−0001−7809−7485]], Abdennour Azerine[1 [0000−0003−2460−3140]], Mahmoud Golabi[1 [0000−0002−3314−2051]], and Lhassane Idoumghar[1 [0000−0001−8853−3968]]

Université de Haute-Alsace, IRIMAS UR 7499, F-68100 Mulhouse, France
{firstname.lastname}@uha.fr

Abstract. This paper addresses the unrelated parallel-machine scheduling problem with non-resumable maintenance under a makespan minimization objective. Maintenance periods restrict machine availability and create a strong coupling between job assignment and sequencing decisions, making solution evaluation computationally expensive. We propose a decomposition-based hybrid genetic algorithm in which the evolutionary search determines the assignment of jobs to machines, thereby defining for each machine the subset of jobs it must process. Given this assignment, each machine-level subproblem is solved optimally using a pseudo-polynomial dynamic programming procedure formulated as a 0–1 knapsack problem. While this exact evaluation ensures accuracy, it constitutes the main computational bottleneck. To reduce this cost, we introduce a surrogate-assisted evaluation mechanism that approximates makespan values and selectively replaces exact computations during the search. The surrogate models are trained online and updated to reflect the evolving search region. Computational experiments on benchmark instances show that the proposed approach achieves substantial runtime reductions while preserving high solution quality. These results demonstrate the effectiveness of combining exact decomposition with data-driven approximation for maintenance-constrained unrelated parallel-machine scheduling.

Keywords: Unrelated parallel machine scheduling · maintenance constraints · genetic algorithm · dynamic programming · surrogate models.

1 Introduction

Scheduling allocates jobs to limited processing resources over time under operational constraints, with the objective of optimizing one or more performance criteria [16]. In manufacturing and production systems, this problem frequently arises in parallel machine environments, where multiple machines operate concurrently, and jobs must be distributed among them [4]. Depending on machine characteristics, such environments are typically classified as identical, uniform,

or unrelated machines, with the latter representing the most general case in which processing times depend on both the job and the selected machine [16]. In these settings, minimizing the makespan—the completion time of the last job—remains one of the most widely studied objectives due to its direct impact on throughput and resource utilization [16].

Classical scheduling models generally assume continuous machine availability. In practice, however, machines are subject to preventive maintenance periods that interrupt production [12]. When maintenance is non-resumable, jobs must be processed entirely before or after the unavailability [22]. This constraint partitions the time horizon into disjoint availability intervals and fundamentally alters the structure of the scheduling problem. For a given machine, the decision of which jobs can be completed before maintenance becomes a capacity-constrained selection problem, introducing a knapsack-type structure and eliminating the order-independence property commonly exploited in makespan scheduling [16].

The resulting problem combines two distinct sources of computational complexity. On the one hand, the unrelated parallel machine scheduling problem $Rm||C_{\max}$ is strongly NP-hard. On the other hand, the presence of non-resumable maintenance introduces additional combinatorial structure at the machine level: even in the single-machine case, the problem reduces to the Subset Sum problem and is therefore weakly NP-hard [11]. These results highlight that assignment decisions and maintenance-induced scheduling constraints contribute differently to the overall complexity and must be addressed jointly.

Despite this challenge, most studies on maintenance-constrained scheduling focus on single-machine environments [12]. Research on parallel machines has mostly addressed identical or uniform cases, while the unrelated-machine setting remains underexplored. Kurt and Çetinkaya [10] studied unrelated parallel machines with non-resumable maintenance using a mixed-integer formulation and heuristic. However, exact formulations are known to struggle with larger instances, limiting their applicability to large-scale problems [15].

The problem naturally decomposes into two layers. The first handles job-to-machine assignment, distributing workload across heterogeneous resources. The second addresses per-machine scheduling of assigned jobs within the availability windows imposed by maintenance. For a fixed assignment, the machine-level problem reduces to selecting a subset of jobs that can be processed before maintenance under a capacity constraint, which can be solved optimally using dynamic programming [19]. However, while dynamic programming provides exact solutions for individual machines, it cannot handle the combinatorial explosion induced by assignment decisions across multiple machines.

This observation motivates hybrid approaches that combine global search over assignment decisions with exact resolution of machine-level subproblems. Metaheuristics, and in particular genetic algorithms, have proven effective for exploring large assignment spaces in parallel machine scheduling [1, 21]. Nevertheless, when exact evaluation procedures are embedded within such frameworks, repeated fitness computation becomes the dominant computational cost. To address this issue, surrogate-assisted optimization has been proposed, where

machine learning models approximate expensive evaluation functions and reduce the number of exact computations required [8]. Despite their potential, such approaches remain largely unexplored for maintenance-constrained scheduling on unrelated parallel machines.

We study the unrelated parallel machine scheduling problem with a single non-resumable maintenance interval per machine under makespan minimization. A decomposition-based hybrid framework uses a genetic algorithm (GA) to assign jobs to machines, with each machine-level subproblem solved optimally via a knapsack-based dynamic programming procedure. To reduce repeated exact evaluations, a surrogate-assisted mechanism approximates makespan values during the search, balancing solution quality and computational efficiency while addressing scalability limits of existing methods.

The remainder of the paper is organized as follows. Section 2 defines the problem, Section 3 presents the solution methodology, and Section 4 details the exact evaluation procedure. Section 5 introduces the surrogate-assisted framework, while Sections 6 and 7 present and discuss the computational results. Section 8 concludes the paper.

2 Problem Definition

We consider n independent jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ to be scheduled on m unrelated parallel machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j has a processing time p_{ij} on machine M_i, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. All jobs are available at time zero, must be assigned to exactly one machine, and preemption is not allowed.

Each machine M_i is subject to a single maintenance period $[\tau_{1i}, \tau_{2i}]$, during which it is unavailable for processing. Jobs are non-resumable: any job that cannot complete before τ_{1i} must start no earlier than τ_{2i}. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ denote an assignment vector, where $\sigma_j \in \{1, \ldots, m\}$ indicates the machine to which job J_j is assigned. For a given assignment σ, let $J_i(\sigma) = \{J_j : \sigma_j = i\}$ denote the set of jobs assigned to machine M_i. The makespan on a specific machine M_i is denoted by $C_i^{\max}(\sigma)$. The objective is to determine a feasible assignment σ that minimizes the global makespan, denoted as: $C_{\max}(\sigma) = \max_{i \in \{1,\ldots,m\}} \{C_i^{\max}(\sigma)\}$. Using the standard three-field scheduling notation [6], the problem is denoted by $Rm \mid nr\text{-}maint \mid C_{\max}$.

3 Solution Methodology

We employ a genetic algorithm [7] to guide the search for the job-to-machine assignment, while all maintenance-related considerations are handled within the fitness evaluation.

3.1 Solution encoding

In the genetic algorithm, each solution 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, pro-

viding a direct integer encoding tailored to the $R \mid nr\text{-}maint \mid C_{\max}$ problem. As each job is assigned to exactly one machine, any chromosome σ defines a feasible assignment, with the decoding step guaranteeing feasibility under the non-resumable constraint by scheduling jobs sequentially and delaying them when necessary to start only after the maintenance window.

For each machine M_i and chromosome σ, the associated job set is $J_i(\sigma) = \{J_j \in J \mid \sigma_j = i\}$, where $i = 1, \ldots, m$. Each set $J_i(\sigma)$ is processed in a fixed order, with completion times computed under the maintenance interval $[\tau_{1i}, \tau_{2i}]$ of machine M_i. Whenever a job overlaps the start of maintenance, its processing is delayed until time τ_{2i}, consistent with the non-resumable maintenance constraint. This encoding guarantees feasibility by construction, eliminating repair mechanisms, while the assignment–scheduling separation enables independent per-machine fitness evaluation under the maintenance constraints.

3.2 Algorithm Overview

The GA follows a standard generational scheme with elitism, with parameters determined through preliminary experimentation, comparing tournament and roulette-wheel selection, one-point and uniform crossover, and swap and uniform mutation operators.

According to this configuration, the process begins with an initial population of 70 candidate solutions, each generated by sampling its gene σ_j uniformly from $\{1, \ldots, m\}$ to ensure diversity. For each individual σ, fitness is evaluated by decomposing the problem into m independent single-machine subproblems. For each machine M_i, the algorithm computes a completion time $C^i_{\max}(\sigma)$ within its maintenance interval $[\tau_{1i}, \tau_{2i}]$, and the global makespan is defined as $C_{\max}(\sigma) = \max_i C^i_{\max}(\sigma)$.

Parents are selected using tournament selection and recombined through uniform crossover with probability $p_c = 0.75$ to generate new offspring solutions. Diversity is maintained using swap mutation with probability $p_m = 0.10$ [2]. At each generation, the best individual is preserved through elitism, and its objective value also serves as a reference value in the surrogate-assisted evaluation mechanism described in Section 4.

4 Single-Machine Evaluation

This section formulates the single-machine subproblem for a fixed assignment σ, establishes its knapsack reduction, and presents the exact dynamic programming algorithm.

4.1 Fixed-Assignment Machine Subproblems

For a given machine M_i, let $J_i(\sigma)$ denote the set of assigned jobs, and let $P_i = \{p_{ij} : J_j \in J_i(\sigma)\}$ be the multiset of processing times. The scheduling problem on M_i reduces to deciding which jobs to execute before and after its maintenance interval $[\tau_{1i}, \tau_{2i}]$.

4.2 Optimal Subset Selection via Knapsack Reduction

Proposition 1. *Given a fixed assignment σ and machine M_i, minimizing $C^i_{\max}(\sigma)$ is equivalent to selecting a subset $B_i \subseteq J_i(\sigma)$ of jobs to maximize the total processing time completed before maintenance, subject to the capacity constraint:*

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

Proof. Let $k_i = |J_i(\sigma)|$. Consider machine M_i with its assigned jobs $J_i(\sigma)$ and corresponding processing times P_i. Since all jobs are available at time zero and no setup times are considered, scheduling reduces to deciding which subset of jobs completes before maintenance.

If the total load satisfies $\sum_{p \in P_i} p \leq \tau_{1i}$, all jobs can be processed before maintenance, giving $C^i_{\max}(\sigma) = \sum_{p \in P_i} p$, which is trivially optimal.

Otherwise, let $B_i \subseteq J_i(\sigma)$ denote the subset of jobs completed before τ_{1i}, and the remaining jobs start after τ_{2i}. Since jobs are non-preemptive and available from time zero, they can be scheduled consecutively in any order within these intervals without increasing the completion time. Thus, the completion time is: $C^i_{\max}(\sigma) = \tau_{2i} + \sum_{J_j \in J_i(\sigma) \setminus B_i} p_{ij}$.

Because the total load is fixed, minimizing $C^i_{\max}(\sigma)$ is equivalent to maximizing the total processing time scheduled before maintenance, which is exactly the knapsack problem in (1).

The induced single-machine subproblem is weakly NP-hard [13], as Proposition 1 establishes its equivalence to a 0–1 knapsack problem [14].

4.3 Optimal Evaluation via Dynamic Programming

The exact optimum of (1) is computed via a pseudo-polynomial dynamic programming method with $O(k_i \tau_{1i})$ time and memory complexity [9], summarized in Algorithm 1, returning the best packed value and, when required, the corresponding subset via backtracking.

When τ_{1i} is large, this exact evaluation may be expensive in both time and memory, motivating the surrogate-assisted strategy described in Section 5.

Exact Subproblem Evaluation: Define $L_i(\sigma) = \sum_{J_j \in J_i(\sigma)} p_{ij}$ as the total load on machine M_i, and let $V_i(\sigma)$ denote the optimal value of (1). Consequently,

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

and the overall makespan is given by $C_{\max}(\sigma) = \max_i \{C^i_{\max}(\sigma)\}$.

5 Surrogate-Assisted Evaluation

This section presents the surrogate-assisted mechanism used to approximate makespan values and reduce repeated exact evaluation costs.

5.1 Surrogate Model and Training Strategy

A surrogate-assisted strategy 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 4. The surrogate is trained online during the genetic algorithm using pairs (σ, y), where σ represents the raw integer encoding of an assignment and $y = C_{\max}(\sigma)$ is the corresponding exact makespan.

The integer encoding is used directly as input features to maintain consistency with the genetic representation. 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.

5.2 Evaluation Policy and Safeguards

The surrogate becomes operational after a three-generation initialization period E_0, during which all candidates are exactly evaluated to build the foundational training dataset. After activation, the model predicts makespan values $\tilde{C}_{\max}(\sigma)$ for most candidates and is periodically retrained at ten-generation intervals to reflect the evolving search trajectory.

Exact evaluation is triggered by a dual criterion: any solution whose surrogate estimate falls below $C_{\max}^{\text{best}}$ receives mandatory exact evaluation. In addition, a random exact evaluation is triggered with probability $\rho = 0.1$ to enforce exact evaluation, preserve exploration, and mitigate systematic prediction errors. Each exact evaluation contributes $(\sigma, C_{\max}(\sigma))$ to the training repository, updating the best-known solution when improvements occur, while remaining candidates use the surrogate prediction as fitness.

Algorithm 1 Exact makespan evaluation for a fixed assignment σ

Require: Assignment vector σ, processing times (p_{ij}), maintenance (τ_{1i}, τ_{2i}) for all machines

for $i \leftarrow 1$ to m **do**
  $\mathcal{P}_i \leftarrow \{p_{ij} : \sigma_j = i\}$
  $L_i \leftarrow \sum_{p \in \mathcal{P}_i} p$
  if $L_i \leq \tau_{1i}$ **then**
    $C_{\max}^i \leftarrow L_i$
  else
    $V_i \leftarrow \textsc{KnapsackDP}(\mathcal{P}_i, \tau_{1i})$
    $C_{\max}^i \leftarrow \tau_{2i} + (L_i - V_i)$
  end if
end for
return $\max_i\{C_{\max}^i\}$

5.3 Surrogate Regression Models

Five surrogate models are evaluated: Random Forests (RF) [3] and Gradient Boosting (GB) [5] for piecewise interactions; Gaussian Processes (GP) [18] for uncertainty quantification; Support Vector Regression (SVR) [20] for kernel-based regression; and Multi-Layer Perceptrons (MLP) for flexible nonlinear approximation. For each model, the hyperparameters are set to default values, as commonly adopted in the literature [17]. Results are reported in Section 6.

6 Computational Experiments

This section assesses the performance of the proposed GA under both exact and surrogate-assisted fitness evaluation. We compare a baseline genetic algorithm using exact dynamic programming with the surrogate-assisted variant introduced in Section 5, across five regression models, on a set of synthetically generated instances. Experiments were conducted on a workstation equipped with an Intel Xeon W-2265 CPU @ 3.50 GHz, 64 GB RAM, running Windows 11 Pro and Python 3.13.

6.1 Instance generation and maintenance setup

Benchmark instances are defined by a pair (m, n), where m is the number of machines and n the number of jobs, across seven configurations: $(m, n) \in \{(2, 20), (2, 50), (3, 50), (2, 70), (3, 70), (2, 100), (3, 100)\}$. Processing times p_{ij} are sampled independently for each machine-job pair, with 20 instances per configuration.

For each machine M_i, the total hypothetical load $L_i = \sum_{j=1}^{n} p_{ij}$ is computed. The maintenance start time is defined as $\tau_{1i} = \lfloor L_i \alpha \rfloor$, where $\alpha \sim U(0.3, 0.7)$, positioning maintenance within the central portion of the processing horizon. The maintenance duration is sampled uniformly from $[d_{\min}, d_{\max}]$, where $d_{\min} = \max\{p_{\min}, \lfloor 0.15\, L_i \rfloor\}$ and $d_{\max} = \max\{d_{\min} + 1, \lfloor 0.40\, L_i \rfloor\}$, and $\tau_{2i} = \tau_{1i} + \text{duration}$. Scaling parameters with L_i preserve comparable disruption levels across instance sizes while naturally yielding heterogeneous maintenance windows.

Performance comparison Across all configurations reported in Table 1, all surrogate-assisted variants achieve substantial runtime reductions relative to the baseline GA, with GA+GB and GA+GP consistently emerging as the most balanced surrogates. At $n = 20$, GA+GB reduces runtime from 435.70 to 85.40 seconds with a makespan increase of only 20.29 units, while GA+GP achieves a runtime of 82.85 seconds with a makespan of 1910.74. This advantage scales with problem size; at $n = 100$ and $m = 2$, GA+GB and GA+GP reduce runtime to 355.72 and 348.75 seconds, respectively, compared to 2594.10 seconds for the baseline GA.

Table 1: Mean results by configuration. Runtime is reported in seconds.

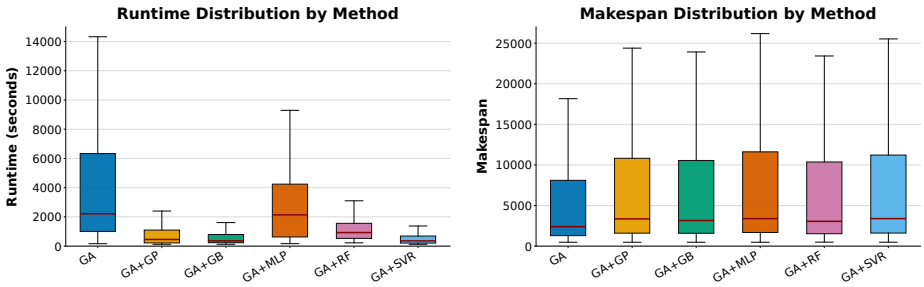
n	m	Method	Makespan	Runtime	Epochs
20	2	GA	1850.21	435.70	124.3
		GA+GP	1910.74	82.85	161.7
		GA+GB	1870.50	85.40	175.4
		GA+MLP	1927.74	1262.56	144.1
		GA+RF	1922.12	297.63	164.6
		GA+SVR	1933.29	73.37	159.5
50	2	GA	4076.21	3164.36	171.3
		GA+GP	4378.81	709.21	316.7
		GA+GB	4554.22	470.72	203.8
		GA+MLP	4959.80	4035.49	190.9
		GA+RF	4560.32	854.18	238.3
		GA+SVR	5068.11	465.21	213.1
50	3	GA	2341.36	4321.69	241.3
		GA+GP	2772.82	1062.15	430.1
		GA+GB	2763.69	592.76	232.8
		GA+MLP	3053.56	3354.68	164.3
		GA+RF	2663.18	1357.50	372.9
		GA+SVR	2923.33	636.63	247.1
70	2	GA	6958.28	7984.61	207.1
		GA+GP	7608.22	1974.78	382.2
		GA+GB	7539.94	1047.22	209.7
		GA+MLP	8580.25	5514.51	164.4
		GA+RF	7448.25	1888.72	315.8
		GA+SVR	8418.94	1197.02	268.2
70	3	GA	3043.72	10729.39	276.0
		GA+GP	4581.17	1367.62	378.0
		GA+GB	4403.97	966.15	182.8
		GA+MLP	5100.56	4522.27	145.4
		GA+RF	4159.68	2281.25	400.4
		GA+SVR	4595.97	1409.12	276.9
100	2	GA	12432.42	2594.10	78.2
		GA+GP	15611.40	348.75	65.5
		GA+GB	14039.77	355.72	75.1
		GA+MLP	16020.90	484.98	41.9
		GA+RF	14412.43	735.57	89.6
		GA+SVR	15943.81	234.44	55.0
100	3	GA	4212.29	9166.66	209.3
		GA+GP	6602.39	1291.16	216.1
		GA+GB	6144.85	848.69	118.0
		GA+MLP	6575.95	2986.03	87.7
		GA+RF	5975.68	1864.21	210.6
		GA+SVR	6570.04	1053.83	130.1

7 Results and Discussion

This section compares the baseline GA against five surrogate-assisted variants. The analysis covers computational efficiency, solution quality, and surrogate prediction accuracy across all tested configurations.

As shown in Figure 1, replacing exact dynamic programming with a surrogate model consistently reduces runtime across all five models, bringing worst-case GA runtimes, over 6000 s, down to a markedly lower and more stable range, with GA+GB and GA+GP performing best (Figure 1a), at the cost of only a modest increase in $C_{\max}$, where GA+RF and GA+GP remain closest to the baseline (Figure 1b).

Surrogate Accuracy Across Problem Scales: The surrogate models remain highly accurate across all problem sizes, with the best reaching mean accuracies near 0.9 in some instances, as shown in Figure 2. Among them, GA+GP, GA+RF, and GA+GB consistently deliver the highest accuracy across all job counts.



(a) Runtime Distribution by Method (b) Makespan Distribution by Method

Fig. 1: Runtime and Makespan Distribution Across Surrogate Methods.

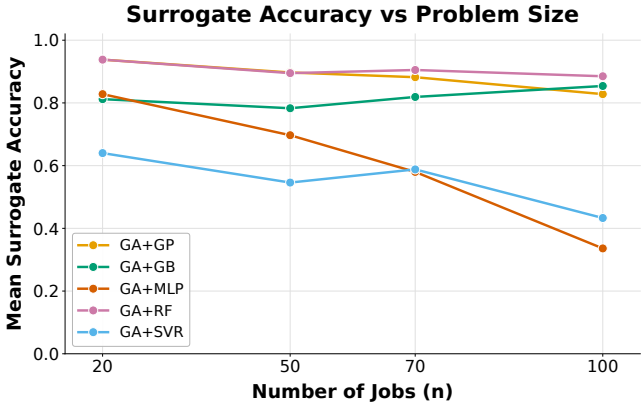
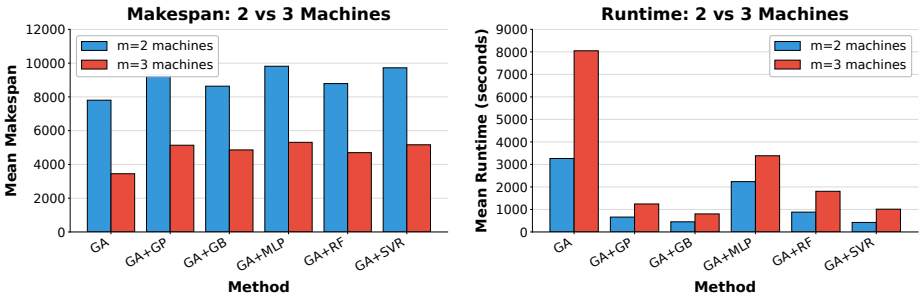


Fig. 2: Surrogate Accuracy.

Effect of Problem Scale: Machine Count Increasing machines from $m = 2$ to $m = 3$, improves $C_{\max}$ across all methods while surrogate-assisted variants remain computationally efficient, as shown in Figure 3. GA+GB stays well below 1000 seconds for both configurations, while the baseline GA climbs to nearly 8000 seconds, underlining the practical value of surrogate assistance as the assignment space expands.

Impact on makespan and solution quality: For small to medium instances n = 20 and n = 50, all surrogate methods yield values of the makespan $C_{\max}$ that are almost indistinguishable from those of the baseline GA, so the quality–efficiency trade-off remains highly favorable; beyond (n = 50), the methods start to diverge, with GA+RF and GA+GP staying closest to the baseline and GA+MLP showing the most noticeable degradation as shown in Figure 4.



(a) Effect of machine count on makespan (b) Effect of machine count on runtime

Fig. 3: Effect of Machine Count on Makespan and Runtime

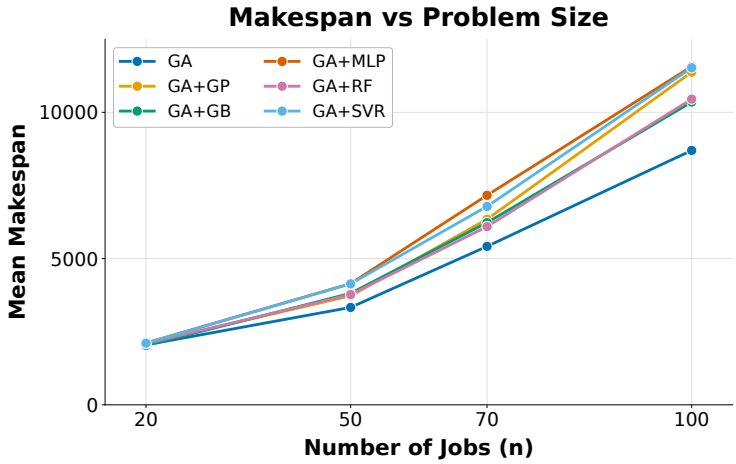


Fig. 4: Effect of Problem Size on Solution Quality Across Surrogate Methods.

8 Conclusion and future perspectives

This paper addressed the unrelated parallel-machine scheduling problem with one non-resumable maintenance window per machine under makespan minimization. To tackle the high computational cost of exact fitness evaluation, we proposed a hybrid framework combining a genetic algorithm for job-to-machine assignment with exact dynamic programming for machine-level subproblem resolution, augmented by a surrogate-assisted evaluation framework benchmarking five regression models: Gaussian Process, Gradient Boosting, Multilayer Perceptron, Random Forest, and Support Vector Regression. The experiments show that all surrogate-assisted variants substantially reduce runtime compared to the baseline GA, with GA+GB and GA+GP consistently emerging as the most balanced in terms of solution quality and computational efficiency. GA+RF and GA+GP maintain the highest prediction accuracy across all problem sizes,

while GA+MLP and GA+SVR degrade at larger scales. Adding a third machine consistently improves $C_{\max}$ across all methods, and surrogate-assisted variants, particularly GA+GB, handle the added complexity with far smaller runtime increases than the baseline GA.

This work opens several promising directions for future research. Adaptive surrogate strategies that dynamically balance exact and approximate evaluations based on search progress could enhance solution quality without sacrificing computational efficiency. Extending the framework to multiple or resumable maintenance periods would introduce richer problem structures and provide a more demanding test for surrogate models. Incorporating local search operators, such as job reassignment following surrogate-guided exploration, could further refine schedule quality. Finally, applying the framework to identical and uniform parallel machine settings would enable assessment of its generalizability across a broader class of scheduling problems.

References

1. Adan, J.: A hybrid genetic algorithm for parallel machine scheduling with setup times: A comparative study of metaheuristics on large problem instances. Journal of Intelligent Manufacturing **33**(7), 2059–2073 (2022)
2. Azerine, A., Golabi, M., Oulamara, A., Idoumghar, L.: Enhancing electric vehicle charging schedules: A surrogate-assisted approach. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 183–186 (2024)
3. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
4. Cheng, T., Sin, C.: A state-of-the-art review of parallel-machine scheduling research. European Journal of Operational Research **47**(3), 271–292 (1990)
5. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of Statistics **29**(5), 1189–1232 (2001)
6. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics **5**, 287–326 (1979)
7. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
8. Ikonen, T.J., Heljanko, K., Harjunkoski, I.: Surrogate-based optimization of a periodic rescheduling algorithm. AIChE Journal **68**(6), e17656 (2022)
9. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004). https://doi.org/10.1007/978-3-540-24777-7
10. Kurt, A., Çetinkaya, F.C.: 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 **15**(1), 18–33 (2024)
11. Lee, C.Y.: Machine scheduling with an availability constraint. Journal of global optimization **9**(3), 395–416 (1996)
12. Low, C., Ji, M., Hsu, C.J., Su, C.T.: Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. Applied Mathematical Modelling **34**(2), 334–342 (2010)
13. Luo, W., Chen, L., Zhang, G.: Approximation algorithms for scheduling with a variable machine maintenance. In: International Conference on Algorithmic Applications in Management. pp. 209–219. Springer (2010)

14. Luo, W., Liu, F.: On single-machine scheduling with workload-dependent maintenance duration. Omega **68**, 119–122 (2017)
15. Pacheco, J., Ángel-Bello, F., Álvarez, A.: A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times. Journal of Scheduling **16**(6), 661–673 (2013)
16. Pinedo, M.L.: Scheduling, vol. 29. Springer (2012)
17. Prieto, N.V., Garrido-Merchán, E.C.: Default machine learning hyperparameters do not provide informative initialization for bayesian optimization. arXiv preprint arXiv:2602.08774 (2026)
18. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
19. Sadfi, C., Penz, B., Rapine, C.: 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)
20. Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. Statistics and computing **14**(3), 199–222 (2004)
21. Vallada, E., Ruiz, R.: A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European Journal of Operational Research **211**(3), 612–622 (2011)
22. Zou, J., Yuan, J.: Single-machine scheduling with maintenance activities and rejection. Discrete Optimization **38**, 100609 (2020)