Improved Modeling and Genetic Algorithm for Job Sequencing and Tool Switching on Non-Identical Parallel Machines

Khaled Khayati
LR-OASIS-ENIT, L@bISEN-Usine du Futur
UTM, ISEN Yncréa Ouest
Tunis-Tunisia, Nantes-France
khaled.khayati@etudiant-enit.utm.tn

Khadija Hadj Salem
Department of Computer Science, NUMA
KU Leuven
Ghent, Belgium
khadijaalice.hadjsalem@kuleuven.be

Benoit Lardeux
L@bISEN, Usine du Futur
ISEN Yncréa Ouest
Nantes, France
benoit.lardeux@isen-ouest.yncrea.fr

Safa Bhar Layeb
Indus. Engineering Center, IMT Mines Albi
Toulouse University
Albi, France
safa.layeb@mines-albi.fr

Zied Jemai
LR-OASIS, ENIT
UTM
Tunis, Tunisia
zied.jemai@enit.utm.tn

Maher Jridi
L@bISEN, Vision-AD
ISEN Yncréa Ouest
Nantes, France
maher.jridi@isen-ouest.yncrea.fr

***Abstract*—The job sequencing and tool switching problem with non-identical parallel machines (SSP-NPM) is a challenging combinatorial optimization problem that arises in flexible manufacturing systems. It jointly involves job assignment, sequencing, and tool management decisions under limited magazine capacities to minimize the makespan. In this paper, we first propose an improved position-based MILP (IPM). This model builds upon the one presented by [1], offering a more compact representation of completion times and is strengthened by static improvement mechanisms inspired by [2]. We then develop a genetic algorithm (GA) using an explicit permutation-assignment encoding and a tool-switch evaluator. Computational experiments were conducted on benchmark instances from the literature. Results show that the IPM solves 58.43% of instances to optimality, outperforming existing formulations, while on medium-size large-scale instances, the proposed GA converges about 50% faster than existing metaheuristic approaches.**

***Index Terms*—Scheduling, Job sequencing and tool switching, Non-identical parallel machines, Integer linear programming, Genetic algorithm**

I. Introduction

Flexible manufacturing systems are central to modern production, where efficiency depends on scheduling and resource management. In such systems, the limited capacity of tool magazines requires coordinated job sequencing and tool switching. This is further complicated by unrelated parallel machines with machine-dependent processing times, capacities, and switching times.

In this context, we address the job sequencing and tool switching problem with non-identical parallel machines (SSP-NPM). This problem consists of assigning each job to a feasible machine, determining job sequences, and ensuring tool switches under capacity constraints, while minimizing the makespan. SSP-NPM is an extension of the classical single-machine tool switching problem (ToSP), also known as the job sequencing and tool switching problem (SSP), introduced in [3], [4]. The SSP is known to be NP-hard, as proven in [5], and becomes significantly more complex in the parallel-machine setting due to the coupling between assignment, sequencing, and tool management decisions. To deal with the parallel-machine setting, different mathematical models and heuristic methods have been proposed for the SSP-NPM. [6] introduced three main mixed-integer linear programming (MILP) formulations, and [7] proposed a multi-commodity flow formulation. Recently, [2] proposed a position-based arc flow model and two job group based formulations, and different modeling improvement techniques. Regarding heuristics, early constructive heuristics for the SSP-NPM were proposed by [8], while an Iterated Local Search (ILS) approach was later introduced for larger instances by [1]. More recent works have focused on hybrid metaheuristics, including hybrid GA with local search (GALS), in particular the GALS-KTNS (Keep Tool Needed Soonest) variant [9], and a biased random-key GA combined with variable neighborhood descent (BRKGA-VND) [10].

Despite these advances, a gap remains between exact and heuristic approaches. While MILP formulations ensure optimality, they scale poorly. Conversely, high-performing metaheuristics often rely on hybrid approaches that combine GA with LS or VND mechanisms. Investigating new GA models with efficient evaluation is still important for achieving competitive performance and scalability. To address this issue, we propose two approaches: an IPM-MILP and a GA.

The remainder of the paper is organized as follows. Section II presents the problem description. Section III introduces the mathematical formulation. Section IV presents the Genetic Algorithm. Section V reports computational experiments, and Section VI concludes the paper.

II. Problem Description

The SSP-NPM is defined as follows. Let $\mathcal{J} = \{1, \ldots, J\}$ denote the set of jobs, $\mathcal{M} = \{1, \ldots, M\}$ the set of unrelated parallel machines, $\mathcal{T} = \{1, \ldots, T\}$ the set of tools, $\mathcal{K} = \{1, \ldots, J\}$ the set of positions, and $\mathcal{K}^m = \{1, \ldots, \hat{K}^m\}$ denote the set of feasible positions on machine m. All jobs are assumed to be available at time zero. Each job $j \in \mathcal{J}$ can be processed on a subset of feasible machines $\mathcal{M}_j \subseteq \mathcal{M}$, and its processing time p_j^m depends on the selected machine m. Each machine $m \in \mathcal{M}$ is equipped with a tool magazine of limited capacity C^m, where each tool occupies one slot. In addition, each machine is associated with a tool switching time sw^m, incurred whenever a tool is inserted into the magazine. Each job j requires a subset of tools $\mathcal{T}_j \subseteq \mathcal{T}$ that must be simultaneously available in the magazine of the machine assigned to process it. The subset of jobs requiring a given tool t is denoted by $\mathcal{J}_t \subseteq \mathcal{J}$. Due to the limited capacity of the magazine, tool switches may be necessary when processing consecutive jobs on the same machine. A tool switch corresponds to inserting a tool into the magazine and induces a setup time. It is assumed that tool switching operations can only occur between two consecutive jobs, that no switching is allowed during job processing, and that the initial loading of tools does not incur any cost.

The decision process consists of assigning each job to one feasible machine, determining the processing sequence of jobs on each machine, and managing tool loading so that capacity constraints are satisfied at all times. These decisions are strongly interdependent, as job assignments and sequencing influence tool switching requirements and, consequently, the overall schedule duration. The objective is to minimize the makespan $C_{\max}$, defined as the completion time of the last finished job across all machines.

Table I represents an SSP-NPM instance with 2 machines, 5 jobs, and 14 tools, where $C^1 = 6$, $C^2 = 5$, and $sw^1 = sw^2 = 2$. The corresponding solution is shown in Figure 1 where machine m_2 processes jobs (j_1, j_2), while machine m_1 processes jobs (j_4, j_3, j_5) and requires two tool insertions before processing j_5. The resulting schedule has a makespan of $C_{\max} = 14$.

TABLE I
Example instance of SSP-NPM

Jobs	**1**	**2**	**3**	**4**	**5**
Tools	1	3	2	11	7
	7	10	10	12	9
	12	–	4	–	14
$\boldsymbol{p_j^1}$	6	7	6	1	2
$\boldsymbol{p_j^2}$	6	8	9	1	9

III. Improved position-based MILP model

The proposed MILP model, denoted as IPM, improves the PM formulation through a more compact completion-time representation and static reduction and initialization mechanisms. The IPM relies on three sets of decision variables. Binary

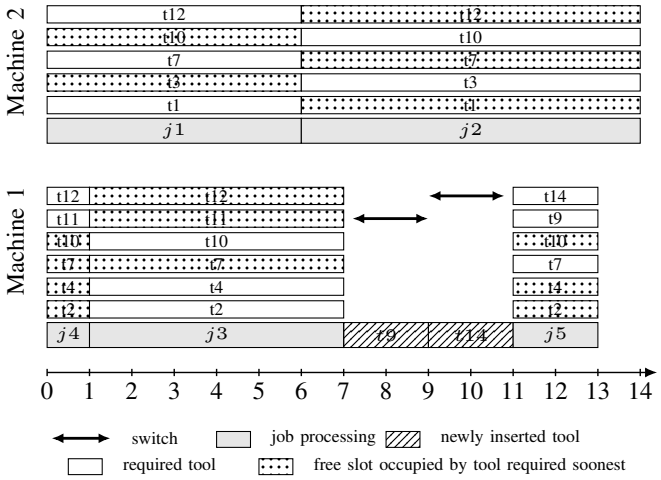

Fig. 1. Illustrative schedule with $C_{\max} = 14$ and TS = 2 in m_1

variables x_{jk}^m indicate the assignment of job j to position k on machine m. Binary variables v_{tk}^m indicate whether tool t is present in the magazine at position k, and u_{tk}^m whether it is inserted before processing. Continuous variables f_k^m denote the completion time of position k on machine m, and $C_{\max}$ the makespan.

$$\min \quad C_{\max} \tag{1}$$

$$\text{s.t.} \quad \sum_{m \in \mathcal{M}} \sum_{k \in \mathcal{K}^m} x_{jk}^m = 1 \quad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_{j \in \mathcal{J}} x_{jk}^m \leq 1 \quad \forall k \in \mathcal{K}^m,\ m \in \mathcal{M} \tag{3}$$

$$\sum_{j \in \mathcal{J}} x_{jk}^m \leq \sum_{j \in \mathcal{J}} x_{j,k-1}^m \quad \forall k \in \mathcal{K}^m \setminus \{1\},\ m \in \mathcal{M} \tag{4}$$

$$x_{jk}^m = 0 \quad \forall j \notin \mathcal{J}(m),\ \forall k \in \mathcal{K}^m,\ m \in \mathcal{M} \tag{5}$$

$$v_{tk}^m \geq x_{jk}^m \quad \forall j \in \mathcal{J}_t,\ \forall k \in \mathcal{K}^m,\ m \in \mathcal{M},\ t \in \mathcal{T} \tag{6}$$

$$\sum_{t \in \mathcal{T}} v_{tk}^m \leq C^m \quad \forall k \in \mathcal{K}^m,\ m \in \mathcal{M} \tag{7}$$

$$u_{tk}^m \geq v_{tk}^m - v_{t,k-1}^m \quad \forall k \in \mathcal{K}^m \setminus \{1\},\ m \in \mathcal{M},\ t \in \mathcal{T} \tag{8}$$

$$u_{t1}^m = 0 \quad \forall t \in \mathcal{T},\ m \in \mathcal{M} \tag{9}$$

$$f_1^m = \sum_{j \in \mathcal{J}} p_j^m \cdot x_{j1}^m \quad \forall m \in \mathcal{M} \tag{10}$$

$$f_k^m \geq f_{k-1}^m + sw^m \cdot \sum_{t \in \mathcal{T}} u_{tk}^m + \sum_{j \in \mathcal{J}} p_j^m \cdot x_{jk}^m \quad \forall k \in \mathcal{K}^m \setminus \{1\},\ m \in \mathcal{M} \tag{11}$$

$$C_{\max} \geq f_k^m \quad \forall k \in \mathcal{K}^m,\ m \in \mathcal{M} \tag{12}$$

In this IPM model, the objective minimizes the makespan $C_{\max}$. Constraints (2)–(12) define the core formulation. Constraint (2) assigns each job exactly once. Constraints (3)–(4) ensure position feasibility and contiguity. Constraint (5) enforces machine compatibility. Constraints (6)–(7) guaran-

tee tool availability and respect magazine capacity. Constraints (8)–(9) model tool insertions with free initial loading. Constraints (10)–(11) define completion-time propagation, and Constraint (12) links all completion times to the makespan.

To strengthen the formulation, we add the following constraints:

$$C_{\max} \geq \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{J}} \min_m p_j^m \tag{13}$$

$$\sum_{j \in \mathcal{J}} \sum_{k=1}^{\lceil \hat{K}^m/2 \rceil} j \cdot x_{jk}^m \geq \sum_{j \in \mathcal{J}} \sum_{k=\lceil \hat{K}^m/2 \rceil + 1}^{\hat{K}^m} j \cdot x_{jk}^m \quad \forall m \in \mathcal{M} \tag{14}$$

Constraint (13) introduces an additional workload-based lower bound by distributing the $\min_m p_j^m$ over all machines. Constraint (14), originally introduced in [2], is incorporated into the proposed IPM formulation to further strengthen the model through symmetry breaking, thereby avoiding redundant exploration of equivalent schedules.

A key difference with respect to the PM of [8] lies in the definition of completion-time variables. While the PM uses job-position variables f_{jk}^m, the IPM relies on position-level variables f_k^m. Since each position contains at most one job, these variables are sufficient, leading to a more compact formulation and improved computational efficiency.

To improve tractability, we adopt reduction and initialization mechanisms from [2]. Unlike their dynamic integration in the JG and IP models of [2], they are applied here statically to provide good initial bounds, thereby improving computational efficiency. A machine-dependent position limit $\hat{K}^m$ is computed for each machine, and positions $k > \hat{K}^m$ are removed. A threshold $\hat{\Theta}(m)$ is estimated to strengthen the makespan bound. A greedy heuristic generates an initial feasible solution used to build an MIP start. These mechanisms are computed once before optimization and remain fixed during the solution process.

IV. Genetic Algorithm

We propose a GA to efficiently explore the large combinatorial space induced by joint job sequencing and machine assignment decisions. In contrast to existing GA-based approaches [9], [10], which rely on hybrid metaheuristics combining GA with LS components such as LS, VND, or BRKGA-based mechanisms, the proposed method uses a compact chromosome (π, a) encoding only sequencing and assignment, while tool feasibility is handled by a dedicated evaluator. This separation simplifies the search process while preserving solution quality. Figure 3 summarizes the overall process of the proposed GA, whose main components are detailed in the following subsections.

A. Chromosome representation

Each individual is defined as: $x = (\pi, a)$ where π is a global permutation of jobs and a is a feasible job-to-machine assignment ($a_j \in M_j$).

Permutation π (priority list)

7	2	5	1	6	3	4

Assignment map a (job $\rightarrow$ machine)

j	1	2	3	4	5	6	7
a_j	2	1	3	2	1	3	1

Decoding: scan π and append job j to σ_{a_j}.

Fig. 2. Chromosome representation based on permutation and assignment

This representation encodes sequencing and assignment decisions, while tool feasibility is handled during evaluation.

B. Decoding

Decoding transforms (π, a) into a schedule $S = \{\sigma_m\}$ by scanning jobs in the order of π and appending each job j to σ_{a_j}. Infeasible assignments are penalized.

C. Fitness function

The fitness of individuals is defined as

$$f(x) = C_{\max}(S(x)) \tag{15}$$

where $S(x)$ is the decoded schedule and $C_{\max}$ is computed via the tool-switch simulation.

D. Schedule Evaluator

Evaluation of a schedule S is performed by simulating, for each machine m, the sequence σ_m under C^m and sw^m.

1) Magazine state: At each position k, the magazine contains a set $A_k^m \subseteq T$ such that

$$|A_k^m| \leq C^m \tag{16}$$

2) Initial loading and prefill: Let $j = \sigma_m(1)$. The initial magazine is

$$A_1^m = T_j \cup F^m \tag{17}$$

where F^m is an optional prefill set such that $|T_j \cup F^m| \leq C^m$. This initial loading is free ($s_1^m = 0$).

Prefill follows an *earliest-next-use* policy.

3) Insertions for subsequent jobs: For $k \geq 2$, required insertions are

$$\Delta_{m,k}^+ = T_j \setminus A_{k-1}^m \tag{18}$$

To satisfy capacity

$$|A_{k-1}^m| - |R_k^m| + |\Delta_{m,k}^+| \leq C^m \tag{19}$$

Tools are removed using a *farthest-next-use* rule.

The insertion time is

$$s_k^m = sw^m \cdot |\Delta_{m,k}^+|, \quad k \geq 2 \tag{20}$$

4) Completion time:

$$f^m(S) = \sum_{k=1}^{|\sigma_m|} p_{\sigma_m(k),m} + \sum_{k=2}^{|\sigma_m|} s_k^m \tag{21}$$

The objective is

$$C_{\max} = \max_{m \in \mathcal{M}} f^m(S) \tag{22}$$

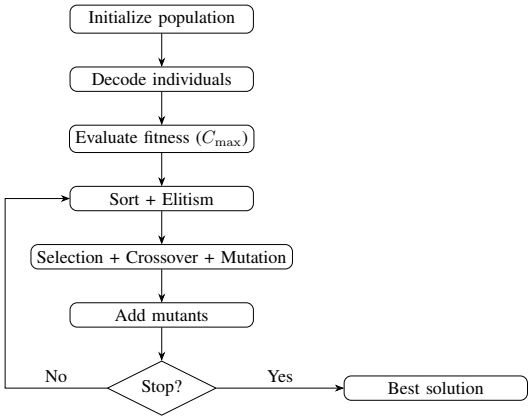

Fig. 3. Evolutionary process of the proposed GA

E. Evolutionary process

Once individuals are represented and evaluated, the evolutionary process is applied as described below. At each generation, the population evolves through a sequence of selection, recombination, and diversification steps. Individuals are first ranked according to their fitness, and a fraction α_e of the best solutions is preserved (elitism). The remaining individuals are generated using tournament selection, crossover, and mutation, while a fraction α_m of the population is replaced by random mutants to maintain diversity.

1) Tournament selection: To select a parent, k individuals are sampled uniformly at random from the population, and the one with the best fitness is chosen. This mechanism ensures a balance between selection pressure and diversity.

2) Crossover operators: Two complementary crossover operators are applied:

a) Order crossover (OX) on π: A subsequence is copied from the first parent, while the remaining positions are filled following the order of the second parent, ensuring a valid permutation.

b) Uniform crossover on a: For each job j, the assignment a_j is inherited from one of the parents with equal probability, followed by a feasibility repair if necessary.

3) Mutation: Two mutation mechanisms are used:

a) Swap mutation: Two positions in the permutation π are randomly exchanged.

b) Reassignment mutation: A job is reassigned to another feasible machine with a small probability.

These operators jointly ensure exploration of the search space while preserving solution feasibility.

V. Computational Experiments

In this section, we analyze the performance of the proposed approaches under a common experimental setting. The exact methods considered are the proposed IPM, the PM of [6], and the JG of [2]. The metaheuristic approaches include the proposed GA, the GALS-KTNS of [9], the ILS of [1], and the BRKGA+VND of [10]. All methods considered in this study were implemented in Python 3.11. The MILP models were solved using the Gurobi Optimizer version 12.0.3. The computational experiments were performed on a PC equipped with an 11th-generation Intel Core i7-1165G7 processor (4 physical cores, 8 logical processors, base frequency of 2.80 GHz). The system was configured with 16 GB of RAM and operated under the Windows 11 environment. A time limit of 3600 seconds was imposed for the MILP models, while 600 seconds was used for the metaheuristic approaches.

A. Instances description

The computational study is based on publicly available benchmark instances from the literature [11], corresponding to the SSP-NPM dataset.

a) Set-I: The first set of instances, referred to as Set-I, contains 140 instances grouped into 7 subsets of 20 instances each. The instances are characterized by a number of machines varying in $\{2, 3\}$, a number of jobs in $\{5, 10, 15, 25\}$, and a number of tools in $\{10, 15\}$. The job-tool incidence matrices are dense (d) and sparse (s), allowing the evaluation of different structural levels of tool requirements.

b) Set-II: The second set, denoted as Set-II, comprises 160 instances divided into 8 subsets of 20 instances each. In this set, the number of machines takes values in $\{2, 3\}$, the number of jobs varies in $\{10, 15, 20\}$, and the number of tools in $\{10, 15, 20\}$.

c) Set-III: The third set, denoted as Set-III, comprises 720 instances divided into 18 subsets of 40 instances each. In this set, the number of machines takes values in $\{2, 3\}$, the number of jobs varies in $\{10, 15, 20\}$, and the number of tools in $\{10, 15, 20\}$.

d) Set-IV: The fourth set, denoted as Set-IV, comprises 480 large-scale instances divided into 6 subsets of 80 instances each publicly available at https://git.io/Jvsqp. In this set, the number of machines takes values in $\{4, 6\}$, the number of jobs varies in $\{40, 60, 120\}$, and the number of tools in $\{60, 120\}$.

B. Numerical results

To analyze the numerical results, we distinguish between the evaluation of exact methods and metaheuristic approaches, each assessed according to specific performance criteria.

1) Mathematical models: For the comparison of mathematical formulations (PM, JG, and the proposed IPM), the analysis is structured along three main axes. First, we evaluate the ability of each model to solve instances to optimality within the time limit by comparing the number of optimally solved instances and the corresponding average CPU time computed over these instances, as reported in Table II . Second, we use a performance profile based on the cumulative percentage of optimal solutions obtained over time (Figure 4). Finally, we complement this analysis with a performance profile based on the cumulative percentage of instances with respect to the Gurobi optimality gap (Figure 5), providing a more detailed comparison of solution quality. Experiments were conducted on Sets I, II, and III. The results indicate that the proposed IPM formulation solves significantly more instances to optimality than PM and JG within the same time limit.

TABLE II
COMPARISON OF MATHEMATICAL MODELS ON SET-I, II AND III

Set	M	J	T	PM [6] CPU	PM [6] #opt	JG [2] CPU	JG [2] #opt	IPM CPU	IPM #opt
	2	5	10	0.16	20	0.21	20	**0.04**	**20**
	2	5	15	0.52	20	0.37	20	**0.08**	**20**
	2	10	10	284.61	20	16.96	20	**2.82**	**20**
	2	10	15	1146.54	18	2021.09	11	**18.35**	**20**
Set-I	2	15	15	–	0	**833.14**	**3**	1349.77	2
	3	15	15	–	0	493.99	7	**974.06**	**11**
	3	25	15	–	0	–	0	–	0
	2	10	10	966.20	20	125.81	20	**14.95**	**20**
	2	10	15	1587.35	18	713.00	16	**98.08**	**20**
	2	15	10	–	0	840.07	15	**758.88**	**15**
Set-II	2	15	15	–	0	–	0	**863.40**	**1**
	3	15	15	–	0	926.38	3	**1184.01**	**14**
	3	20	15	–	0	–	0	–	0
	3	15	20	–	0	–	0	**2979.70**	**1**
	3	20	20	–	0	–	0	–	0
	2	10	10	2176.14	28	39.60	40	**10.72**	**40**
	2	10	15	2163.69	22	120.72	39	**23.08**	**40**
	2	10	20	2780.53	6	304.09	37	**30.99**	**40**
	2	15	10	–	0	**139.99**	**38**	274.98	37
	2	15	15	–	0	641.78	16	**613.50**	**27**
	2	15	20	–	0	461.93	5	**1231.98**	**10**
Set-III	2	20	10	–	0	**444.74**	**32**	453.19	12
	2	20	15	–	0	**802.48**	**4**	872.88	2
	2	20	20	–	0	**32.56**	**3**	495.72	3
	3	10	10	1220.41	31	14.04	40	**6.66**	**40**
	3	10	15	1726.95	21	186.04	40	**17.92**	**40**
	3	10	20	2446.31	12	345.03	39	**33.00**	**40**
	3	15	10	–	0	341.66	37	**264.47**	**37**
	3	15	15	–	0	624.51	15	**562.49**	**25**
	3	15	20	–	0	368.85	3	**1302.98**	**16**
	3	20	10	–	0	**642.09**	**27**	736.29	17
	3	20	15	–	0	**2125.09**	**6**	1635.35	5
	3	20	20	–	0	**204.37**	**3**	94.74	1
Total #opt					236		559		**596**

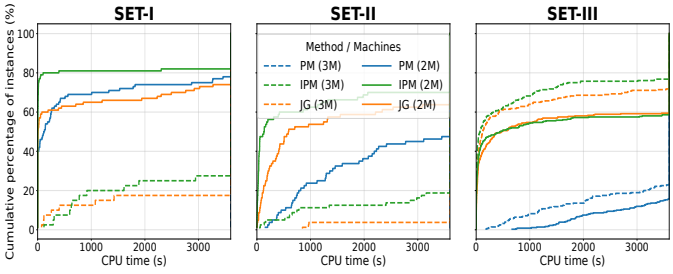

Fig. 4. Performance evaluation based on CPU time for Set-I, II, and III

As shown in Figure 4, the analysis distinguishes between instances with 2 and 3 machines. In both cases, IPM consistently dominates PM and JG, reaching higher cumulative percentages within shorter CPU times. As illustrated in Figure 5, and consistently with the previous analysis, IPM achieves lower optimality gaps for a large proportion of instances, with results reported separately for instances with 2 and 3 machines. For two-machine instances, IPM achieves more than 90% of instances with an optimality gap below 0.2, while for three-machine instances, at least 50% of instances remain below this threshold. This highlights the ability of IPM to provide stronger lower bounds, thereby improving the convergence of the branch-and-bound process. Additionally, IPM and PM consistently provide feasible solutions for all instances (reaching 100% coverage), while JG exhibits a slightly lower coverage.

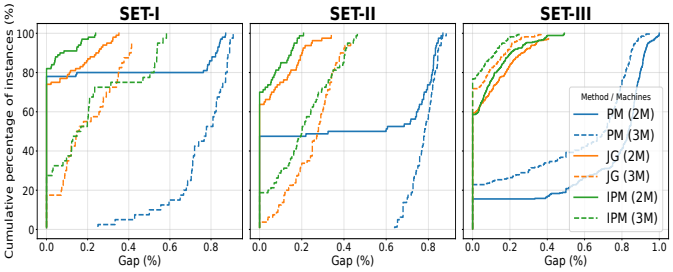

Fig. 5. Performance evaluation based on optimality gap for Set-I, II, and III

2) Metaheuristic approaches: For the metaheuristic approaches, namely GALS-KTNS [9], BRKGA+VND [10], ILS [1], and the proposed GA, the analysis focuses on solution quality evaluated through the percentage deviation from the best solution obtained among all methods. More precisely, for each instance, the deviation, expressed as a percentage gap ($Gap(\%)$), is computed as:

$$Gap(\%) = \frac{C_{\max}^{\text{method}} - C_{\max}^{\text{best}}}{C_{\max}^{\text{best}}} \times 100 \tag{23}$$

where $C_{\max}^{\text{best}}$ denotes the best makespan obtained over all considered methods. The analysis is conducted on Set-IV. All metaheuristic methods were executed using the parameter settings reported in their respective references to ensure a fair comparison. For the proposed GA, the parameter configuration is inspired by [10] and adapted to improve convergence efficiency. The population size is defined as $P = \max(8|J|, 60)$, with an elite fraction $\alpha_e = 0.15$ and a mutant fraction $\alpha_m = 0.25$. Parent selection is performed using tournament selection with size $k = 3$, combined with a crossover rate of 0.90. Mutation is applied with a rate of 0.25, while the reassignment rate is set to 0.10 to enhance diversification in the assignment decisions, which are critical in the SSP-NPM problem. A fixed random seed ($seed = 123$) is used to ensure reproducibility. In contrast to existing approaches such as [10] and [9], which rely on hybridization with VND and LS, respectively, the proposed approach is based solely on evolutionary mechanisms. The parameter configuration is therefore designed to balance exploration and exploitation within the GA, ensuring effective convergence without the need for hybrid components. Table III reports the average deviation ($Av_{Gap(\%)}$) for each instance subset.

The results highlight two distinct behaviors depending on the number of machines. For the 4-machine subsets, the proposed GA achieves the lowest average deviation ($Av_{Gap(\%)} = 1.07$), outperforming GALS-KTNS (3.23), BRKGA+VND (4.47), and ILS (35.28). This confirms that, for medium-size large-scale instances, the proposed representation and evaluation scheme provide an effective balance between assignment and sequencing decisions. For the 6-machine subsets, GALS-KTNS clearly dominates, with an average deviation of 0.75, compared with 8.03 for the proposed GA, 11.08

TABLE III
COMPARISON OF METAHEURISTIC METHODS ON SET-IV (600S)

Set	M	J	T	ILS [1] $Av_{Gap(\%)}$	BRKGA+VND [10] $Av_{Gap(\%)}$	GALS-KTNS [9] $Av_{Gap(\%)}$	GA $Av_{Gap(\%)}$
	4	40	60	38.09	4.74	**0.74**	2.30
	4	60	60	37.90	6.60	7.44	**0.20**
Set-IV	4	40	120	31.35	2.27	**0.90**	1.55
	4	60	120	33.79	4.27	3.84	**0.22**
	6	80	120	32.55	6.56	**1.30**	5.31
	6	120	120	29.33	15.60	**0.19**	10.74

for BRKGA+VND, and 30.94 for ILS. This indicates that, as the number of machines increases, the LS-based hybrid intensification becomes more effective than an evolutionary approach. The larger assignment space and the stronger interaction between sequencing and tool loading favor methods with dedicated improvement mechanisms.

3) Convergence behavior: To further analyze the search dynamics, the convergence of the two best-performing methods, GALS-KTNS and the proposed GA, is compared for instances with 4 and 6 machines (Figures 6, 7). The curves report the evolution of the median normalized gap over generations.

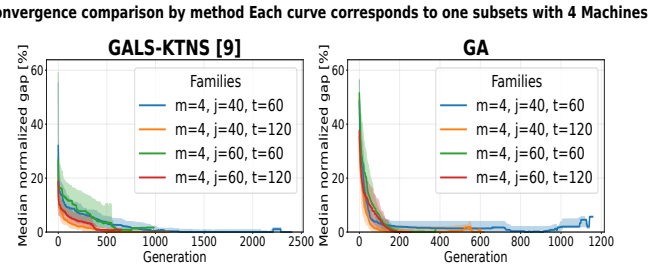

Fig. 6. Convergence comparison on 4-machine subsets

For the 4-machine subsets (Figure 6), the proposed GA converges faster on most instance families, with a sharp gap reduction and stabilization before 200 generations, whereas GALS-KTNS exhibits a more gradual convergence, often extending beyond 400 generations. This observation is consistent with Table III, where the proposed GA achieves the best average performance, due to the moderate assignment space that enables the assignment-sequencing encoding to quickly identify effective solutions while tool switching is handled by the evaluator. For the 6-machine subsets (Figure 7),

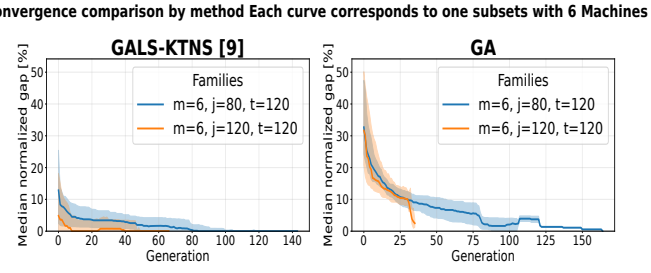

Fig. 7. Convergence comparison on 6-machine subsets

GALS-KTNS converges faster and attains lower gaps, with early stabilization of the curves, whereas the proposed GA improves more gradually and remains at higher gap levels. This behavior is explained by the larger assignment space and the stronger coupling between assignment, sequencing, and tool availability, where GALS-KTNS-based intensification enables deeper solution refinement.

VI. CONCLUSION

In this paper, we addressed the SSP-NPM with the objective of minimizing the makespan. We proposed an IMP MILP and a GA based on an explicit assignment-sequencing representation coupled with a tool-switch evaluator. The IPM relies on a more compact completion-time representation and static reduction mechanisms, while the GA separates the main combinatorial decisions from tool management during evaluation.

Computational results show that the proposed IPM improves the performance of exact solution methods and solves more instances to optimality than the reference formulations. For large-scale instances, the proposed GA achieves competitive performance and proves particularly effective on the 4-machine subsets, while GALS-KTNS remains more effective on the more complex 6-machine subsets. These results confirm the relevance of the proposed exact and metaheuristic approaches for the SSP-NPM. As future work, it would be interesting to extend these approaches to stochastic variants of the SSP-NPM.

REFERENCES

[1] D. Calmels, "An iterated local search procedure for the job sequencing and tool switching problem with non-identical parallel machines," *European Journal of Operational Research*, vol. 297, pp. 66–85, 2022.

[2] K. Hadj Salem, A. Kramer, and A. Robbes, "Job sequencing and tool switching problem with non-identical parallel machines: mathematical formulations and modeling improvements," *European Journal of Operational Research*, 2025.

[3] C. Tang and E. Denardo, "Models arising from a flexible manufacturing machine, part ii: Minimizing the number of tool switches," *Operations Research*, vol. 36, no. 5, pp. 778–784, 1988.

[4] J. F. Bard, "A heuristic for minimizing the number of tool switches on a flexible machine," *IIE Transactions*, vol. 20, no. 4, pp. 382–391, 1988.

[5] Y. Crama, A. W. Kolen, A. G. Oerlemans, and F. C. Spieksma, "Minimizing the number of tool switches on a flexible machine," *IJFMS*, vol. 6, no. 1, pp. 33–54, 1994.

[6] D. Calmels, "A comparison of different mathematical models for the job sequencing and tool switching problem with non-identical parallel machines," *International Journal of Operational Research*, 2020.

[7] T. T. Da-Silva, A. A. Chaves, and H. H. Yanasse, "A new multicommodity flow model for the job sequencing and tool switching problem," *International Journal of Production Research*, 2020.

[8] D. Calmels, C. Rajendran, and H. Ziegler, "Heuristics for solving the job sequencing and tool switching problem with non-identical parallel machines," in *Operations Research Proceedings 2018*. Springer International Publishing, 2019, pp. 459–465.

[9] T. Cura, "Hybridizing local searching with genetic algorithms for the job sequencing and tool switching problem with non-identical parallel machines," *Expert Systems with Applications*, vol. 223, p. 119908, 2023.

[10] L. C. R. Soares and M. A. M. Carvalho, "Biased random-key genetic algorithm for the job sequencing and tool switching problem with non-identical parallel machines," *Computers & Operations Research*, vol. 163, p. 106509, 2024.

[11] K. A. Hadj Salem, A. Kramer, and A. Robbes, "Instances for the job sequencing and tool switching problem with non-identical parallel machines (ssp-npm)," 2025.