Annales Mathematicae et Informaticae
61 (2025) pp. 94–107
DOI: 10.33039/ami.2025.10.011
URL: https://ami.uni-eszterhazy.hu

Multi-objective genetic and memetic algorithms in flexible flowshop scheduling

Levente Áron Fazekas[a], **Károly Nehéz**[b]

[a]Institute of Information Technology, University of Miskolc
levente.fazekas@uni-miskolc.hu
karoly.nehez@uni-miskolc.hu

Abstract. Flowshop scheduling problems are classic examples of scheduling where the objective is to minimize makespan – the total manufacturing time. Only the processing times required for each operation are considered. Since the general flowshop problem is NP-hard, multiple heuristic and metaheuristic approaches have emerged over the years. Most practical applications, however, require a much more nuanced approach: multiple – sometimes contradictory – objectives must be considered simultaneously alongside a plethora of additional constraints. Flexible flowshop problems are an abstraction of classic flowshops, where each stage can consist of multiple parallel machines, referred to as work centres. Commonly, models also have to consider a broader range of manufacturing restrictions and variables, such as setup times, machine eligibility restrictions, and due dates. This study aims to demonstrate the application of genetic and memetic metaheuristic algorithms on the $FFc \mid s_{i,j,k},\ d_j,\ M_j \mid C_{max},\ T_{max},\ \sum_j T_j,\ \sum_j U_j$ flexible flowshop scheduling problem. It also outlines a dynamic decoding method for permutation or random key representations to alleviate controllability and tightness problems during genotype-phenotype conversion. Common genetic crossover and mutation operations are showcased alongside the simulated annealing local search algorithm to form memetic algorithms. To handle multiple objectives, a modified version of the relative distance method is employed. The findings are demonstrated via the Taillard benchmark set.

Keywords: genetic algorithm, memetic algorithm, simulated annealing, multi-objective scheduling, flowshop scheduling.

AMS Subject Classification: 90B35, 90C23

1. Introduction

Flowshop scheduling problems are classic examples of constrained multi-resource and multi-operation scheduling [12]. Classically, the objective is to minimize make-

Accepted: October 8, 2025
Published online: October 28, 2025

span – the total manufacturing time – where only the processing times required for each operation are considered. Since the general flowshop problem is NP-hard, multiple heuristic and metaheuristic approaches have emerged over the years. Most practical applications, however, require a much more nuanced approach: multiple – sometimes contradictory – objectives must be considered simultaneously alongside a plethora of additional constraints. Flexible flowshop problems [10, 20] are an abstraction of classic flowshops, where each stage can consist of multiple parallel machines, referred to as work centres. Commonly, models also have to consider a broader range of manufacturing restrictions and variables, such as setup times, machine eligibility restrictions, and due dates [21, 23]. This study aims to demonstrate the application of genetic and memetic metaheuristic algorithms on the $FFc \mid s_{i,j,k},\ d_j,\ M_j \mid C_{max},\ T_{max},\ \sum_j T_j,\ \sum_j U_j$ flexible flowshop scheduling problem. It also outlines a dynamic decoding method [29] for permutation or random key representations [26] to alleviate controllability and tightness problems during genotype-phenotype conversion. Common genetic crossover and mutation operations are showcased alongside the simulated annealing local search algorithm to form memetic algorithms. To handle multiple objectives, a modified version of the relative distance method [14, 15] is employed as opposed to common weighted sum or ε-constraint methods [2] or non-dominating sorting genetic algorithms [5, 19]. The findings are demonstrated via the Taillard benchmark set [24], where the original problem has been transformed to accommodate the denoted problem.

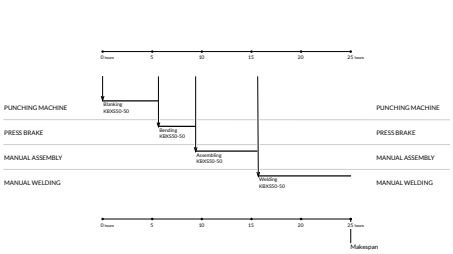
Most flexible flowshop studies choose to focus on throughput-related performance indicators, such as minimizing makespan or flow time. In particular cases, throughput may not be the most important or the only objective. In make-to-order manufacturing, a late order implies a penalty in the form of loss of goodwill, and the magnitude of the penalty depends on the tardiness of the delivery [20]. In many circumstances, managing on-time delivery has significance alongside improving the system's throughput. Optimizing due-date-related schedule metrics, such as the number of tardy orders, total tardiness, and maximum lateness, is crucial for manufacturing firms.

In the literature, algorithms for solving the flexible flowshop problem can be categorized into exact and heuristic approaches. Exact methods, including mathematical programming and branch and bound, create optimal solutions. Due to the lack of efficient lower bounds, the branch and bound approach is limited to simple shop configurations. Exact methods require a long time for solving large instances. Both facts limit the practical application of these methods. A more practical method is to search for a quasi-optimal solution in a reasonable amount of time. For this reason, the trend is to solve flexible flowshop problems using heuristics, especially metaheuristics.

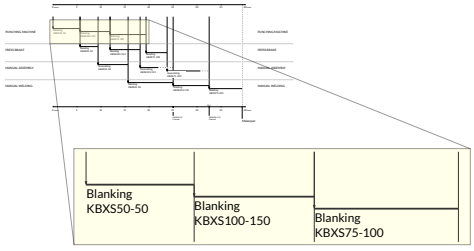
2. Problem formulation

A scheduling problem can be described by a triplet $\alpha|\beta|\gamma$ notation [23]. The α field describes a machine environment and usually contains just one entry. The β field

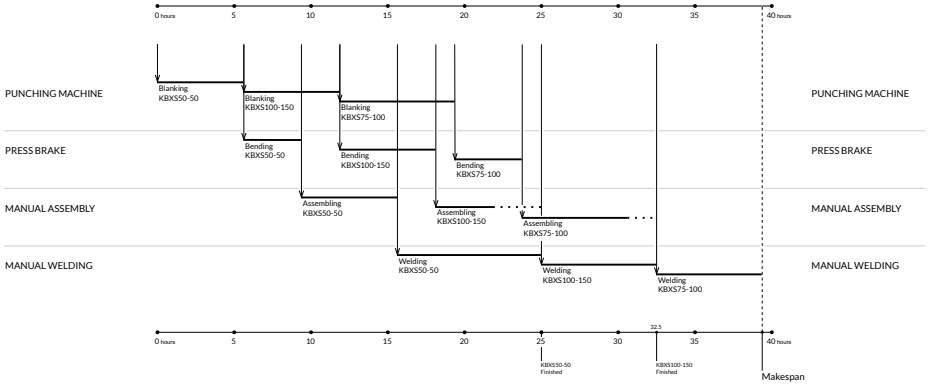
details the processing characteristics and constraints of operations. This β field may contain multiple, single, or no entries. The γ field describes the performance metric or metrics to be minimized.



(a) An example of the first job contingency in the classic flowshop problem.



(b) An example of first machine contingency in the classic flowshop problem.



(c) An example of the flow-shop problem.

Figure 1. A classic flowshop problem.

Consider a flowshop F environment with n jobs and m machines: $p_{j,i}$ represent the processing times. $i \in \{1,\ 2,\ \ldots,\ m\}$ is the ith machine in the production line, $S_{j,i}$ is the starting time of job j on machine i, $C_{j,i}$ is the completion time of job j on machine i. A resource can only process one job at a time; therefore, the start time of the next job must be equal to or greater than the completion time of its predecessor on the same resource. A job can only be present on one resource at any given time, meaning that the start time of the same job on the next machine must be greater than or equal to the completion time of its predecessor operation.

$$C_{j,i} \leq S_{j+1,i}$$
$$C_{j,i} \geq C_{j,i+1}$$
$$C_{1,i} = \sum_{k=1}^{i} p_{1,k}$$

$$C_{j,1} = \sum_{l=1}^{j} p_{j,l}$$
$$C_{j,i} = \max(C_{i-1,j}, C_{i,j-1}) + p_{i,j}$$

The first scheduled job does not have to wait for other jobs and is available from the start $S_{1,1} = 0$. The completion time of the first job on the current machine is always the sum of its previous operations on the preceding machine in the chain and its processing time on the current machine. Figure 1a shows how the first scheduled job has only one contingency - its operations on previous machines. Jobs on the first machine are only contingent on jobs on the same resource. Therefore, the completion time of the job on the first machine is the sum of previously scheduled jobs plus its own and $S_{i,1} = C_{i-1,1}$. Figure 1b illustrates that the first machine has only one contingency, and operations can follow without delay. Considering subsequent jobs on subsequent machines $(i, j > 1)$, the completion times are contingent on the same job on previous machines and previously scheduled jobs on the previous machines in the chain. Figure 1c illustrates the solution for a classic flowshop problem using a Gantt chart.

The classic flowshop problem aims to minimize the completion time of the last job called the makespan. Therefore, the aim is to minimize the completion time of the last scheduled machine on the last machine in the manufacturing line:

$$C_{max} = C_{n,m} \to \min$$

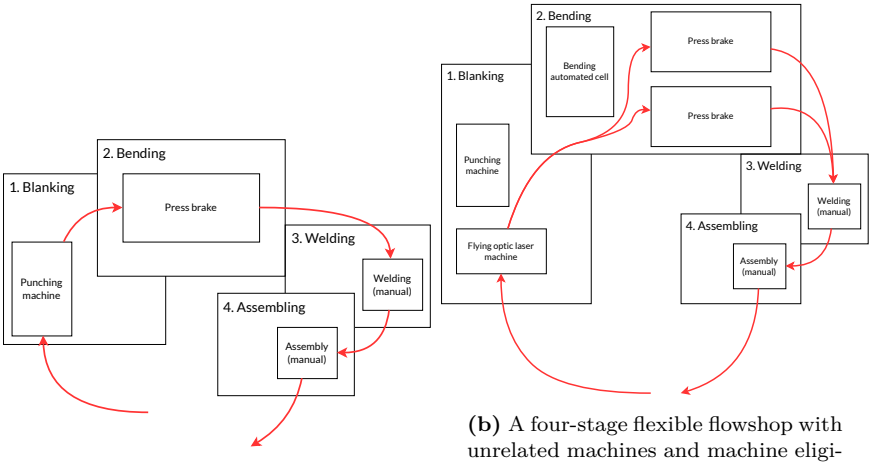
2.1. A flexible flowshop example

A flexible flowshop is a generalization of the classic flowshop (Fm) and parallel machine (Pm) environments. Instead of m machines in series, there are c stages with several identical machines in parallel. Each job must be processed first at stage 1, then stage 2, and so on. A stage functions as a bank of parallel machines; at each stage, job j requires processing on only one machine, and any machine can do it. The queues between the various stages may or may not operate according to the *First Come First Served (FCFS)* principle. In literature, flexible flowshops have also been referred to as hybrid and multiprocessor flowshops. The following flexible flowshop problem is presented as an example:

$$FFc \mid s_{i,j,k},\ d_j,\ M_j \mid C_{max},\ T_{max},\ \sum_j T_j,\ \sum_j U_j$$

In this paper, we present a problem with identical parallel machines at each stage (FFc), machine eligibility constraints (M_j), sequence-dependent setup times $(s_{i,j,k})$, due dates (d_j), and multiple objective functions. Machine eligibility indicates that not all machines can process any job in a stage due to certain limitations – this characteristic is significant in the modern industry but rarely considered by the literature [23].

For example, a four-stage flexible flowshop, a sheet metal manufacturing environment, is shown. The system consists of four stages: blanking, bending, welding, and assembling. Metal sheets enter the blanking stage, cutting the raw material into two-dimensional parts with a laser cutting machine or punching press. Then, the parts are transferred to the bending stage to be bent into specific three-dimensional parts. After bending, the welding and assembling stages take the parts to a completed - final product - state. In the cutting stage, a laser cutting machine may not be able to be used for all types of materials. Bending requires specific tool sets and work ranges, which may rule out specific machines for a particular part. Figure 2b illustrates such a four-stage flexible flowshop environment and a possible path in the system as opposed to the classic flowshop example illustrated in Figure 2a.



(a) A classic flowshop environment.

(b) A four-stage flexible flowshop with unrelated machines and machine eligibility.

Figure 2. Comparison of Fm and FFc models.

3. Encoding and decoding methods

Encoding is a representation of a solution by a vector of values representing key decision variables on which an algorithm operates. This vector is often called a genotype or chromosome in case of genetic algorithms. Most generally, schedules are represented by each operation's start and finish times on every corresponding machine. This view allows for an infinite solution space. Since scheduling often involves minimizing makespan, flowtime, and lateness, all operations are commonly started as early as possible. This goal makes the schedule a semi-active schedule [20], in which no operation can be completed earlier without change the processing order. In such schedules, the decision variables are reduced to the machine assignment of each operation and the sequence of operations on each machine. A large encoding scheme in a large-scale solution may result in inefficient searching. Urlings

et al. [26] studied different encoding schemes in genetic algorithms and deducted that more detailed encodings result in worse scheduling performance. For this reason, most use a permutation encoding scheme, where a solution is represented as a job order $s \in \mathcal{S}_n$. Alternatively, random keys can replace direct permutations by mapping real valued vectors $s \in [0,1)^n$ to job orders via sorting.

Decoding derives a schedule (phenotype) from the encoded solution (genotype). Permutation encoding does not contain all necessary decision variables when constructing a flexible flowshop schedule. These missing variables, like machine selection, are determined by heuristics during decoding; for this reason, decoding methods are crucial to solution quality. The most adopted method is List Scheduling (LS), where jobs are executed in the order given in the encoding on the first stage and using the first-come-first-served (FCFS) principle on all subsequent ones. With List Scheduling, the original ordering's influence on the schedule is diminished by the FCFS rule through the various stages. This limited influence is known as the *controllability problem*. Another widely used method is Permutation Scheduling (PS), as adopted by Ruiz and Stützle [22]. As opposed to List Scheduling, Permutation Scheduling keeps the initial, global ordering for all stages. This method improves control, making it easier to handle urgent jobs, but it can cause idle time when stages desynchronize. This inefficiency is called the *tightness problem*.

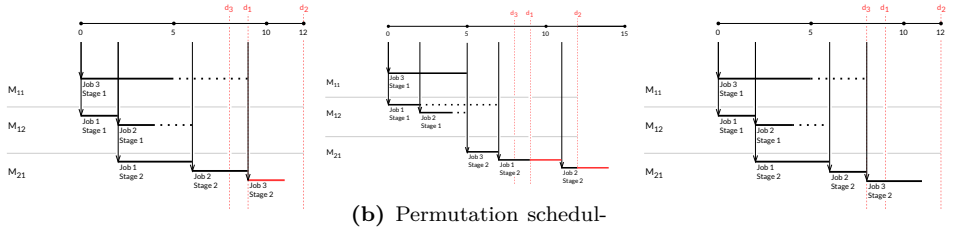
Despite their broad application, permutation and list scheduling both have obvious drawbacks. In scheduling, one may want to handle urgent jobs without the delay caused by synchronization. For this reason, Chunlong et al. [29] utilized permutation encoding and *Dynamic Scheduling* with *first available* machine selection to minimize the total tardiness $\sum_j T_j$ while maintaining tightness and controllability. In Dynamic Scheduling, both the completion time on the previous machine $C_{j,i-1}$ and the global order s are used. When a machine finishes a job, it chooses from jobs available at that time, but in the order given by the initial order s. This method combines List Scheduling and Permutation Scheduling by modifying the machine's buffer into a priority-queue.

Table 1. A 2-stage scheduling problem.

Job	Stage 1 Eligibility	Processing time	Stage 2 Eligibility	Processing time	Due
1	$\{M_{1,1}, M_{1,2}\}$	2	$\{M_{2,1}\}$	4	9
2	$\{M_{1,1}, M_{1,2}\}$	2	$\{M_{2,1}\}$	3	12
3	$\{M_{1,1}, M_{1,2}\}$	5	$\{M_{2,1}\}$	2	8

Table 1 presents a two-stage flowshop problem as an illustrative example. Suppose the solution is given by the Earliest Due Date (EDD) heuristic, resulting in the job sequence $s = \{3, 1, 2\}$. List Scheduling is depicted in Figure 3a. Due to differing completion times in stage 1, the job order in stage 2 changes to $s = \{1, 2, 3\}$. The resulting performance indicators are: makespan $C_{\max} = 11$, total tardiness $\sum_j T_j = 3$, maximum tardiness $T_{\max} = 3$, and number of late jobs $\sum_j U_j = 1$ (Job 3). Figure 3b shows the outcome of applying permutation-based scheduling.

To maintain the original sequence s across both stages, the start times of Jobs 1 and 2 in stage 2 are delayed. The resulting objective values are: $C_{\max} = 14$, $\sum_j T_j = 4$, $T_{\max} = 2$, and $\sum_j U_j = 2$. In contrast, Figure 3c illustrates the result of Dynamic Scheduling. Here, when Job 1 completes stage 1, it immediately proceeds to stage 2 and is assigned to machine $M_{2,1}$. Later, when $M_{2,1}$ becomes available, both Job 2 and Job 3 are waiting. Since Job 2 has higher priority according to s, it is selected for processing before Job 3. This dynamic adjustment achieves the same makespan as List Scheduling, $C_{\max} = 11$, but crucially, it was able to uphold Job 2's priority, eliminating all tardiness: $\sum_j T_j = 0$.



(a) List-scheduling. **(b)** Permutation scheduling. **(c)** Dynamic scheduling.

Figure 3. Different decoding methods.

4. Examined algorithms

This paper focuses on three prominent methods: Simulated Annealing (SA), Genetic Algorithms (GA), and Memetic Algorithms (MA).

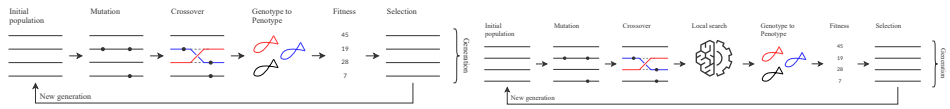
Simulated Annealing (SA), introduced by Kirkpatrick et al. [13] and Cerny [3], is a probabilistic variant of hill climbing that accepts worsening moves with decreasing probability. Inspired by thermodynamic cooling, it explores a cost landscape via an inhomogeneous Markov chain. The cooling schedule $T(t)$ is a monotonic function impacting convergence quality [16]. For benchmarking, Multiplicative Exponential cooling was used.

Genetic algorithms are search algorithms that mimic natural selection and genetics [11]. The Swap2, Swap3, Adjacent, Reverse segment, Shift segment, Shuffle segment mutations were considered. *Recombination operators* combine parts of two parent solutions to generate one or more offspring. The following operators were considered: Order 1 Crossover (OX1) [4, 9], Order 2 Crossover (OX2) [7, 9], Partially Mapped Crossover (PMX) [8, 9], Cycle Crossover (CX) [18], Edge Recombination Crossover [27].

Selection mechanisms fall into fitness-proportionate (e.g., roulette) and ordinal (e.g., tournament, truncation) categories [7].

Memetic Algorithms (MA) enhance GAs by integrating local search. Inspired by Moscato's model [17], they apply local refinement to globally guided search, improving both quality and convergence speed [25]. Hybrids with nested SA as inner search and MA as outer search have demonstrated superior approximation

performance [1, 6, 28]. Figures 4a and 4b illustrate the core steps of GA and MA, respectively.



(a) Genetic algorithm. **(b)** Memetic algorithm.

Figure 4. Comparison of memetic and genetic algorithms.

For multi-objective optimization a modified relative distance method was used [14]. The method utilizes two feasible solutions simultaneously and generates a numeric value based on their relative distances. This value is derived from K fitness function value pairs, which are compared and scaled. These scaled values are then multiplied by a weight that signifies an objective's importance. The resulting scaled and multiplied values are then summed. The signedness of the sum signifies dominance or equality.

$$\begin{gathered} F\colon S^2 \to \mathbb{R} \\ F(s_x, s_y) = \sum_{i=1}^{k} D(s_x, s_y) \\ D(s_x, s_y) = \begin{cases} \dfrac{f_i(s_y) - f_i(s_x)}{\max(f_i(s_x), f_i(s_y))} & \text{if } \max(f_i(s_x), f_i(s_y)) \neq 0, \\ 0 & \text{otherwise} \end{cases} \end{gathered} \tag{4.1}$$

Our modified version uses an *ideal* point $f_i^* = \min\{f_i(s) : s \in P\}$ to derive a distance for every individual in a population $\mathcal{P}(\mathcal{S})$ and sorts them accordingly – the closer an individual is to the ideal, the better. The D function is also made to use the total of absolute values of each objective function value. Therefore, our $F\colon S \times \mathcal{P}(S) \to \mathbb{R}$ and $D\colon S \times \mathcal{P}(S) \to \mathbb{R}$ can be applied to an entire population. Equation (4.2) formulates our modified approach as opposed to the original method detailed in Equation (4.1).

$$\begin{gathered} F(s_x, P) = \sum_{i=1}^{k} D(s_x, P) \\ D(s_x, P) = \begin{cases} \dfrac{f_i^* - f_i(s_x)}{|f_i^*| + |f_i(s_x)|} & \text{if } |f_i^*| + |f_i(s_x)| \neq 0, \\ 0 & \text{otherwise} \end{cases} \end{gathered} \tag{4.2}$$

5. Benchmark sets

The basis of all benchmarks was the dataset published by Taillard [24]. To generate machine configurations (work centres), constants from the set $1, 2, 4$ were selected,

resulting in two scenarios: a *standard case*, where all stages have the same number of machines, and a *bottleneck case*, where the last stage contains only a single machine, simulating a bottleneck in the production flow. Notably, when all stages consist of a single machine, the problem reduces to the classical flowshop variant.

Due dates were generated with a looseness factor $l = 1.3$, using the upper bound UB of each problem as a reference. The due date bounds were computed as $d_{lb} = 0.75 \cdot UB \cdot l$ and $d_{ub} = 1.25 \cdot UB \cdot l$. Each due date d_j was then sampled uniformly as an integer from the interval $[d_{lb}, d_{ub}]$. Setup times $s_{i,j,k}$ were generated randomly as integers in the range $[0, 20]$. For simplicity, total eligibility was assumed – i.e., all jobs are eligible for all machines at every stage.

All random generation was performed using C++'s `std::mt19937` pseudo-random number generator, initialized with the original seed used by Taillard.

6. Results

All mutation, crossover combinations were run on all problems in the 20 job, 5 machine Taillard set (`1tai20_5`). All algorithms were run 10 times – totaling 8100 runs – with the following parameters:

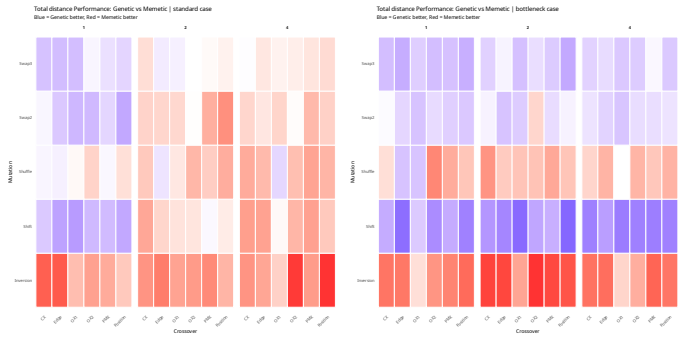
Table 2. Benchmarking parameters.

Parameter	Value
GA generations	1000
MA generations	100
Chromosome count (GA & MA)	20
SA iteration count (MA)	100
SA initial temperature	3000
SA α	0.1

Distance values were calculated similarly to Equation (4.2), using the entire result set from all algorithms on a specific problem, stage, bottleneck configuration. Figure 5 clearly shows how memetic algorithms edge out genetic algorithms on standard benchmarks and Inversion mutation was the operator that benefitted the most from the introduction of a local search algorithm.

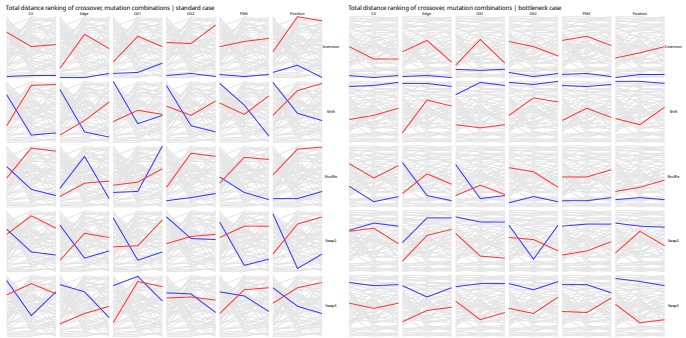
A percentile difference ω from the best known solution is also used. The measure for comparison is the upper bound, the best solution known so far. From the obtained final makespan C_{BS} and the upper bound C_{UB}, a difference is calculated $\omega = \frac{C_{BS} - CUB}{C_{UB}} \cdot 100$.

Table 3 presents the best configurations for Genetic Algorithms (GA) and Memetic Algorithms (MA) across varying numbers of machines per stage and bottleneck settings. Configurations were selected based on the lowest total distance from ideal objective values. For each condition, the GA and MA configurations are compared side by side to highlight their relative performance.



(a) Heatmap on the standard case.

(b) Heatmap on the bottleneck case.



(c) Ranking slopegraphs for the standard case.

(d) Ranking slopegraphs for the bottleneck case.

Figure 5. Relative distance comparison per mutation, recombination for each machine count on standard and bottleneck cases.

In the case of one machine per stage without bottlenecking, the Memetic Algorithm using the Inversion/CX combination outperforms its genetic counterpart in all metrics. It achieves a lower makespan (C_{max}), reduced total tardiness ($\sum T$), fewer late jobs ($\sum U$), and a lower ω value, although the genetic configuration with Shift/OX1 remains competitive with a marginally better total distance. Under the same machine configuration but with a bottleneck present, the Genetic Algorithm demonstrates superior performance across all evaluated objectives. The configuration using Shift/Position achieves the best results, with minimal C_{max}, tardiness, and late jobs, as well as the lowest ω and total distance, indicating GA's adaptability in constrained environments at this scale.

With two machines per stage and no bottleneck, both algorithms perform near optimally. The Genetic Algorithm with the Swap3/OX1 configuration achieves perfect scheduling, indicated by zero tardiness and no late jobs. The Memetic Algorithm also reaches optimality with a nearly identical performance, showing both methods to be equally effective under these conditions.

Table 3. Best configurations by total distance per stage.

Alg.	M[1]	B[2]	C_{max}	$\sum T$	$\sum U$	ω	F	Config[3]
GA	1	F	1712.82	302.31	3.06	40.70	**0.01**	Shift/OX1
MA	1	F	**1636.28**	**245.25**	**2.43**	**34.23**	0.02	Inversion/CX
GA	1	T	**1693.31**	**254.16**	**2.43**	**38.76**	**0.01**	Shift/Position
MA	1	T	1723.82	282.51	3.06	41.50	0.02	Shuffle/CX
GA	2	F	**981.94**	**0.00**	0.00	-19.39	**0.01**	Swap3/OX1
MA	2	F	985.26	**0.00**	0.00	**-19.18**	**0.01**	Inversion/Position
GA	2	T	**1379.80**	**42.93**	**0.63**	**13.05**	**0.18**	Shift/Position
MA	2	T	1389.63	69.48	0.72	13.72	0.19	Shift/OX2
GA	4	F	609.87	**0.00**	**0.00**	-49.98	**0.01**	Shuffle/OX1
MA	4	F	**607.59**	**0.00**	**0.00**	**-50.19**	**0.01**	Shuffle/Position
GA	4	T	**1356.70**	56.88	**0.36**	**10.95**	**0.39**	Shift/OX2
MA	4	T	1377.07	**30.60**	0.54	12.79	0.40	Swap2/Edge

When a bottleneck is introduced at the same stage count, the Genetic Algorithm once again proves more robust. Its Shift/Position configuration yields better results in every objective, outperforming the memetic counterpart and reinforcing GA's advantage under more constrained and complex processing.

At four machines per stage with no bottleneck, the Memetic Algorithm takes the lead. The configuration with Shuffle/Position outperforms GA's best setup by achieving slightly better values for C_{max}, ω, and total distance, suggesting that memetic search strategies scale more effectively with increased stage parallelism.

Finally, in the most complex case – four machines per stage with a bottleneck – both algorithms show strengths in different aspects. The Genetic Algorithm configuration yields a better makespan and fewer late jobs, while the Memetic Algorithm significantly reduces total tardiness and produces a more favorable ω value. Although the Genetic Algorithm has a marginally better total distance, the overall results indicate a trade-off between the two strategies depending on the specific scheduling objective.

These findings suggest that Genetic Algorithms tend to perform better in low-stage or bottlenecked environments, while Memetic Algorithms excel as system complexity increases and resources are less constrained. The mutation and crossover pairings also play a critical role, with combinations like Shift/Position, Shuffle/OX1, and Inversion/CX consistently appearing among the top-performing configurations.

Selection of a parameter set for Memetic Algorithms must also consider the runtime overhead introduced by the additional local search operation. This creates a balancing act in CPU time-management between local and global search.

[1]Machines per stage
[2]Bottleneck configuration
[3]Mutation/Crossover

7. Conclusion

This study presented a comparative evaluation of Genetic and Memetic Algorithms for solving flexible flowshop scheduling problems under varying machine stage counts and bottleneck conditions. The results indicate that Genetic Algorithms tend to perform better in constrained environments, especially when bottlenecks are present or stage counts are low, while Memetic Algorithms show superior scalability and robustness as the degree of parallelism increases. The evaluation across multiple objective functions – makespan, total tardiness, number of late jobs, and an aggregate ω metric – highlighted the importance of pairing crossover and mutation operators effectively with the problem structure.

8. Future research

These findings are compelling and provides ample room for future research. Future work will explore several extensions of this study, including integrating both fine-grained and coarse-grained parallel algorithms to reduce runtime and improve scalability. Adding realistic constraints such as setup times, machine eligibility, and maintenance. Combining metaheuristics with Constraint Programming could offer better feasibility guarantees. Using neural networks to guide job and machine selection within metaheuristics, enabling adaptive scheduling policies. Extending with algorithms to explore trade-offs between makespan, tardiness, and other objectives.

References

[1] A. Agárdi, K. Nehéz, O. Hornyák, L. T. Kóczy: *A Hybrid Discrete Bacterial Memetic Algorithm with Simulated Annealing for Optimization of the Flow Shop Scheduling Problem*, Symmetry 13.7 (2021), p. 1131, doi: 10.3390/sym13071131.

[2] S. Aghakhani, M. S. Rajabi: *A new hybrid multi-objective scheduling model for hierarchical hub and flexible flow shop problems*, AppliedMath 2.4 (2022), pp. 721–737, doi: 10.3390/appliedmath2040043.

[3] V. Černỳ: *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*, Journal of optimization theory and applications 45 (1985), pp. 41–51, doi: 10.1007/BF00940812.

[4] L. Davis et al.: *Applying adaptive algorithms to epistatic domains.* In: IJCAI, vol. 85, Citeseer, 1985, pp. 162–164, isbn: 0934613028.

[5] K. Deb, M. Ehrgott: *On Generalized Dominance Structures for Multi-Objective Optimization*, Mathematical and Computational Applications 28.5 (2023), issn: 2297-8747, doi: 10.3390/mca28050100, url: https://www.mdpi.com/2297-8747/28/5/100.

[6] L. Fazekas, B. Tüű-Szabó, L. T. Kóczy, O. Hornyák, K. Nehéz: *A Hybrid Discrete Memetic Algorithm for Solving Flow-Shop Scheduling Problems*, Algorithms 16.9 (2023), issn: 1999-4893, doi: 10.3390/a16090406, url: https://www.mdpi.com/1999-4893/16/9/406.

[7] D. E. Goldberg, B. Korb, K. Deb: *Messy genetic algorithms: Motivation, analysis, and first results*, Complex systems 3.5 (1989), pp. 493–530, issn: 0891-2513.

[8] D. E. Goldberg, R. Lingle Jr: *Alleles, Loci, and the Traveling Salesman Problem*, in: Proceedings of the 1st International Conference on Genetic Algorithms, 1985, pp. 154–159, doi: 10.4324/9781315799674.

[9] Y. Guan, Y. Chen, Z. Gan, Z. Zou, W. Ding, H. Zhang, Y. Liu, C. Ouyang: *Hybrid flow-shop scheduling in collaborative manufacturing with a multi-crossover-operator genetic algorithm*, Journal of Industrial Information Integration 36 (2023), p. 100514, issn: 2452-414X, doi: 10.1016/j.jii.2023.100514, url: https://www.sciencedirect.com/science/article/pii/S2452414X23000870.

[10] J. N. Gupta, S. K. Gupta: *Single facility scheduling with nonlinear processing times*, Computers & Industrial Engineering 14.4 (1988), pp. 387–393, issn: 0360-8352, doi: 10.1016/0360-8352(88)90041-1, url: https://www.sciencedirect.com/science/article/pii/0360835288900411.

[11] J. H. Holland: *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT press, 1992, isbn: 9780262275552, doi: 10.7551/mitpress/1090.001.0001.

[12] S. M. Johnson: *Optimal two-and three-stage production schedules with setup times included*, Naval research logistics quarterly 1.1 (1954), pp. 61–68, doi: 10.1002/nav.3800010110.

[13] S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi: *Optimization by simulated annealing*, science 220.4598 (1983), pp. 671–680, doi: 10.1126/science.220.4598.671.

[14] G. Kulcsár, F. Erdélyi: *A new approach to solve multi-objective scheduling and rescheduling tasks*, International Journal of Computational Intelligence Research 3.4 (2007), pp. 343–351.

[15] K. Mihály, G. Kulcsár: *A New Many-Objective Hybrid Method to Solve Scheduling Problems*, International Journal of Industrial Engineering and Management 14.4 (2023), pp. 326–335, doi: 10.24867/IJIEM-2023-4-342.

[16] J. Miliczki, L. Fazekas: *Comparison of Cooling Strategies in Simulated Annealing Algoithms for Flow-shop Scheduling*, Production Systems and Information Engineering 10.3 (2022), pp. 129–136, doi: 10.32968/psaie.2022.3.10.

[17] P. Moscato: *On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms*, Technical Report, Caltech Concurrent Computation Program Report 826, (1989).

[18] I. M. Oliver, D. J. Smith, J. R. C. Holland: *A study of permutation crossover operators on the traveling salesman problem*, in: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, Cambridge, Massachusetts, USA: L. Erlbaum Associates Inc., 1987, pp. 224–230, isbn: 0805801588.

[19] A. Opris, D.-C. Dang, F. Neumann, D. Sudholt: *Runtime Analyses of NSGA-III on Many-Objective Problems*, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '24, Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1596–1604, isbn: 9798400704949, doi: 10.1145/3638529.3654218.

[20] M. L. Pinedo: *Scheduling, Theory, Algorithms, and Systems*, 6th ed., Springer Cham, 2022, isbn: 9783031059216, doi: 10.1007/978-3-031-05921-6.

[21] I. Ribas, R. Leisten, J. M. Framiñan: *Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective*, Computers & Operations Research 37.8 (2010), Operations Research and Data Mining in Biological Systems, pp. 1439–1454, issn: 0305-0548, doi: 10.1016/j.cor.2009.11.001, url: https://www.sciencedirect.com/science/article/pii/S0305054809002883.

[22] R. Ruiz, T. Stützle: *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*, European Journal of Operational Research 177.3 (2007), pp. 2033–2049, issn: 0377-2217, doi: 10.1016/j.ejor.2005.12.009, url: https://www.sciencedirect.com/science/article/pii/S0377221705008507.

[23] R. Ruiz, J. A. Vázquez-Rodríguez: *The hybrid flow shop scheduling problem*, European Journal of Operational Research 205.1 (2010), pp. 1–18, ISSN: 0377-2217, DOI: 10.1016/j.ejor.2009.09.024, URL: https://www.sciencedirect.com/science/article/pii/S0377221709006390.

[24] E. Taillard: *Benchmarks for basic scheduling problems*, European Journal of Operational Research 64.2 (1993), Project Management anf Scheduling, pp. 278–285, ISSN: 0377-2217, DOI: 10.1016/0377-2217(93)90182-M, URL: https://www.sciencedirect.com/science/article/pii/037722179390182M.

[25] B. Tüű-Szabó, P. Földesi, L. T. Kóczy: *An efficient evolutionary metaheuristic for the traveling repairman (minimum latency) problem*, International Journal of Computational Intelligence Systems 13.1 (2020), pp. 781–793, DOI: 10.2991/ijcis.d.200529.001.

[26] T. Urlings, R. Ruiz, F. S. Serifoglu: *Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems*, International Journal of Metaheuristics 1.1 (2010), pp. 30–54, DOI: 10.1504/IJMHeur.2010.033122.

[27] L. D. Whitley, T. Starkweather, D. Fuquay: *Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator*, in: Proceedings of the 3rd International Conference on Genetic Algorithms, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 133–140, ISBN: 1558600663.

[28] A. J. Wilson, D. Pallavi, M. Ramachandran, S. Chinnasamy, S. Sowmiya: *A review on memetic algorithms and its developments*, Electrical and Automation Engineering 1.1 (2022), pp. 7–12, DOI: 10.46632/eae/1/1/2.

[29] C. Yu, Q. Semeraro, A. Matta: *A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility*, Computers & Operations Research 100 (2018), pp. 211–229, ISSN: 0305-0548, DOI: 10.1016/j.cor.2018.07.025, URL: https://www.sciencedirect.com/science/article/pii/S030505481830217X.