Date of publication xxxx 00, 0000, date of current version November 30, 2025.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Graph Coloring Models for Production Line Scheduling Optimization

ALEJANDRO SOSA CORRAL[1], LUCÍA GARCÍA LADO[2]
[1]Master on Artificial Intelligence, Loyola University, Sevilla, Spain (e-mail: asosacorral@al.uloyola.es)
[2]Master on Artificial Intelligence, Loyola University, Sevilla, Spain (e-mail: lgarcialado@al.uloyola.es)

ABSTRACT Environmental impact and energy consumption have become major concerns for manufacturing industries in recent years. Reducing machine activity and optimizing production time can significantly decrease the ecological footprint of factories. To address the energy- and time-aware scheduling problem, we model the task as a graph coloring problem and propose an evolutionary optimization framework based on two genetic algorithms (single-objective and multi-objective), comparing their performance against a simulated annealing baseline. The single-objective genetic algorithm minimizes a weighted sum of penalty terms and performance indicators, while the multi-objective variant explicitly explores the trade-off between peak machine load and schedule feasibility. All methods are evaluated on standard benchmark instances (FT06, FT10, and LA01) from the job-shop scheduling literature. The results show that the single-objective genetic algorithm consistently produces conflict-free schedules and outperforms the other approaches for medium- and large-sized problems, whereas the multi-objective algorithm is more competitive on smaller instances. Although simulated annealing does not achieve fully feasible solutions, it demonstrates promising behavior for larger problem sizes. Overall, the proposed framework demonstrates that graph coloring models combined with evolutionary optimization constitute an effective strategy for environmentally oriented production scheduling.

INDEX TERMS Energy-aware scheduling, Genetic algorithms, Graph coloring problem, Job-shop scheduling, Simulated annealing

I. INTRODUCTION

Production scheduling is a necessity for companies that want to remain competitive in increasingly dynamic markets [1]. As a result, it has become a field of extensive research in order to find effective approaches to this problem. In particular, production lines are the backbone of production and require precise time management. Any delays in the manufacture of a single component can trigger a chain reaction in every company that depends on it.

In recent years, the rise of industry 5.0 [2] has further broadened the scope of production scheduling, introducing not only economic and operational considerations, but also ecological and environmental ones. The environmental footprint caused by manufacturing has become a critical concern, with production lines accounting for 12% of emissions in the US alone [3] due to high energy consumption and inefficient resource allocation. As global industries expand, these inefficiencies compound, amplifying these environmental burdens. Figure (1) [4] shows the gas emissions produced by a single factory.

Addressing these challenges requires the integration of optimization techniques that enable a simultaneous focus on environmental objectives, in addition to economic and operational goals. By incorporating resource allocation and scheduling strategies, we aim to significantly reduce the environmental footprint of manufacturing systems without compromising productivity. This optimization represents not only a technological opportunity but also an environmental necessity.

Thus, the central problem addressed in this paper concerns how to schedule tasks across production lines in a way that optimizes resource utilization while minimizing energy consumption and environmental impact. Traditional scheduling approaches often prioritize productivity metrics, often overlooking ecological concerns. Therefore, there is a need for methods that integrate sustainability directly into production planning.

Specifically, we will cover a graph-theoretic optimization method based on the graph coloring problem to reduce idle times, balance workloads, and minimize energy consump-

FIGURE 1: The environmental footprint caused by production lines has become a critical concern.

tion, ultimately enhancing both operational and environmental performance. [5] [6] [7].

In summary, the main contributions of this paper are:

- ...
- ...
- ...

The rest of the paper will be structured as follows: it will start with Section II, where...

II. RELATED WORKS

Graph coloring has always been considered a fundamental problem in the combinatorial optimization field, often being applied to scheduling problems. The act of assigning colors to graph vertices in a way that no two connected vertices share a color naturally lends itself to many resource allocation and conflict avoidance scenarios. As such, many works have explored algorithms that try to solve the problem itself as well as how they can be applied to diverse scheduling tasks.

The idea of applying the graph coloring problem to these scheduling tasks is not unfounded. In [5] they applied various approaches, such as iterated local search algorithms to the well-known timetable scheduling problem and found that approaching the task as a graph coloring problem was the most efficient way to solve it, causing the least conflicts in allocation. On the same note, [8] and [9] also explored in depth which genetic operators offered the greatest results to solve the graph coloring problem.

The most common scheduling problem tackled by applying graph coloring is **resource allocation**. For example, in [6], by representing tasks as graph vertices and conflicts as edges, graph coloring is used to schedule resource projects efficiently while minimizing conflicts between tasks that require shared resources. This allows for conflict-free schedules with improved resource utilization.

Similarly, this resource allocation approach was applied to cloud computing. [7] investigates how graph coloring can be utilized to dynamically allocate virtual resources to tasks, reducing execution times for most tasks by optimizing task-to-resource assignments under capacity constraints.

In [10] they take it a step further and use graph coloring problem algorithms as an objective function in clustering of jobs with different priorities across multiple shifts, which were subject to setup and capacity constraints. The objective was to minimize the number of urgent clusters (clusters that contain at least one urgent job) and the total number of clusters.

A notable number of recent works focus on **mixed graph coloring problems**, an extension of graph coloring where some edges may be directed. This is due to their applicability to scheduling tasks where not only do some tasks conflict with one another, but some tasks do precede others – two very recent examples being in [1] and [11]. The former article applied mixed graphs to the job-shop scheduling problem (where the operations of each job are completely ordered), proposing an algorithm to minimize both execution and job completion time, while the latter addressed parallel execution environments in the context of multiprocessor task scheduling, where minimizing idle processor time is crucial.

The evolution and recent developments of the mixed graph coloring problem were extensively surveyed by the main author of the previously mentioned multiprocessor task scheduling article in [12], which provides a comprehensive overview of the development of mixed graph coloring techniques and their broad applicability to various resource scheduling contexts.

Lastly, more recent work has explored modern techniques as an approach to graph coloring. [13] introduces a neural network based method for finding minimal-cost colorings, demonstrating promising results on complex graphs, relevant to more contrived scheduling tasks.

Overall, these works illustrate the variety of applications of graph coloring to scheduling problems across multiple domains, reflecting the continued relevance of graph coloring as a tool for addressing scheduling challenges.

III. STATEMENT OF THE PROBLEM

The Graph Coloring Problem is a classical combinatorial optimization problem defined on an undirected graph $G = (V, E)$, where V denotes the set of vertices and $E \subseteq V \times V$ the set of edges connecting pairs of vertices. The objective of the Graph Coloring Problem is to assign a color $d(v) \in D$ from a finite color set D to each vertex $v \in V$ so that no two adjacent vertices share the same color.

$$d(u) \neq d(v), \quad \forall (u, v) \in E.$$

The goal is to minimize the total number of colors $|D|$ required to achieve a valid coloring. This problem is NP-Hard [10]. Figure (2) shows an example of a solved graph coloring problem.

The Graph Scheduling Problem has been found to be equivalent to General Shop Scheduling Problems [11]. The Job Shop Scheduling problem consists of a set of jobs $J = \{J_1, J_2, \ldots, J_s\}$ that must be processed on different machines $M = \{M_1, M_2, \ldots, M_m\}$. Each Job $J_k \in J$ consists of a set of ordered operations $O = \{O_1, O_2, \ldots O_o\}$ that need to be performed on specific machines $M_t \in M$. Two operations of the same job cannot be performed at the

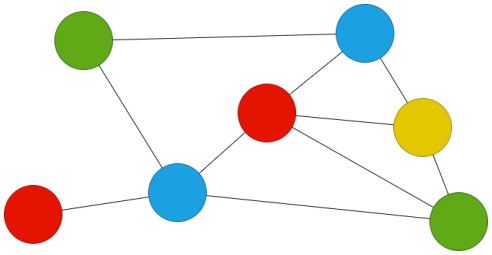

FIGURE 2: Example of a solved graph coloring problem G(7,10).

same time. Machines spend t_n time units (slots) on each operation. To solve this problem, it is necessary to find a schedule that minimizes the completion time $C_{\max} = \max\{C_1, C_2, \ldots, C_s\}$ of the problem while following the two restrictions without generating conflicts. [11]

$$J|t_n|C_{max}$$

To preserve analogy with the graph coloring problem, operations O are modeled as vertex V, task conflicts as edges E, and the minimization objective as colors D.

In this paper, we are going to solve a Job Scheduling problem modeled as a Graph Coloring Problem to minimize the ecological impact of a production line.

A. MAIN OBJECTIVE

To address the environmental concerns of our problem, we consider two possible strategies: minimizing the total operating time of machines and minimizing the number of machines operating simultaneously. Although the second objective may appear counterintuitive, it reduces overall energy consumption by maximizing the utilization of active machines [14]. In addition, this strategy mitigates energy consumption peaks, making it the preferred objective in our formulation. Therefore, we define our problem as follows.

$$\min F(x) = L_{\text{peak}}(x) \tag{1}$$

$$L_{\text{peak}}(x) = \max_{t\in[0,C_{\max}]} |M_t| \tag{2}$$

Where M_t is the set of machines active in time slot t, and $|M_t|$ denotes the cardinality of that set (the number of simultaneously active machines).

B. CONSTRAINTS

In (1) We are only minimizing the number of machines per time slot, however, we need to maintain the same restrictions as the original scheduling problem. This is formulated as constraints. Each constraint represents edges in the graph of the problem.

1) Machine conflict constraint

A single machine can execute only one operation at a time, analogous to a vertex in a graph being connected only with different color vertex. Consequently, our formulation must prevent multiple operations from being scheduled on the same machine within a single time slot. This constraint can be expressed as.

$$P_{\text{mach}} = \sum_{M_k\in M} \sum_{\substack{i,j\in J_{M_k} \\ i<j}} \text{overlap}(i,j) \tag{3}$$

Where J_{M_k} is the set of operations assigned to machine M_k, and overlap(i,j) is an indicator function that returns 1 if operations i and j overlap in time, and 0 otherwise: The overlap function can be mathematically represented by.

$$\text{overlap}(i,j) = \begin{cases} 1 & \text{if } \max(s_i,s_j) < \min(s_i+d_i, s_j+d_j) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Here, s_i, s_j are the start times and d_i, d_j are the durations of operations i and j.

2) Precedence conflict constraint

As exposed previously, operations on each job are ordered. To be able to express this in the Graph Coloring Problem, we need to create a new constraint. It is described by the following expression.

$$P_{\text{prec}} = \sum_{j\in J} \sum_{k=1}^{|O_j|-1} \text{viol}(o_{j,k}, o_{j,k+1}) \tag{5}$$

Where viol checks if a successor operation starts before its predecessor finishes:

$$\text{viol}(u,v) = \begin{cases} 1 & \text{if } s_v < s_u + d_u \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

3) Makespan penalty

If we consider only the number of machines operating per time slot, the total completion time (makespan) may increase uncontrollably, leading to counterproductive outcomes. To mitigate this risk, we introduce a makespan penalty that discourages excessively long schedules. This constraint is defined as follows:

$$C_{\max} = \max(s_s + d_s) \tag{7}$$

Where $C_{\max}$ is the makespan, the completion time of the last operation in the schedule.

C. FINAL PROBLEM DEFINITION

If we combine the function (1) with the constraints (3), (5) and (7) we obtain the following equation.

$$\begin{aligned} \min\ & F(x) = \gamma L_{\text{peak}}(x) + \delta C_{\max} \\ \text{s.t. } & P_{\text{mach}} \approx 0, \\ & P_{\text{prec}} \approx 0, \\ & C_{\max} > 0 \end{aligned} \tag{8}$$

Where $L_{\text{peak}}(x)$ is the maximum number of simultaneous machines (peak load), $C_{\max}$ is the makespan, and P_{mach} and P_{prec} represent the count of machine and precedence conflicts, respectively.

D. ASSUMPTIONS

Assumption 1: In this problem, we assume that reducing energy consumption leads to a corresponding decrease in environmental pollution.

Assumption 2: We assume that all machines consume energy at the same rate. If this were not the case, the total number of machines operating in each time slot would need to be weighted by their respective levels of energy consumption.

$$L_{\text{peak}}(x) = \max_t \left| \sum_{m \in M_t} E_m \right|$$

Where E_m is the energy consumption rate of machine m.

Assumption 3: We assume that every operation takes an exact number of time slots.

Assumption 4: We assume time slots of 1 minute.

Assumption 5: We assume that all machines are available at any time and remain in working condition.

Assumption 6: We assume that no external factors alter the original scheduling problem.

E. DATA SET ACQUISITION

The data set used in this study was obtained from [15]. It contains a collection of Job Scheduling Problem instances compiled from key publications in the scheduling field. In total, it includes 90 problem instances, some of which provide the optimal makespan values. This data set enables an evaluation of our proposed approach in various scheduling scenarios.

In particular, we used the Fisher and Thompson FT106 and FT10 problems and the Lawrence LA01 problem. The FT06 problem contains 6 jobs and 6 machines. The FT10 problem contains 10 jobs and 10 machines. The LA01 problem contains 10 jobs and 5 machines.

IV. PROPOSED EVOLUTIONARY APPROACH

To solve the problem defined in Section III, we propose an evolutionary optimization strategy based on a Genetic Algorithm (GA). Genetic Algorithms are stochastic search heuristics inspired by the natural evolution process [16], [17]. They have been shown to be effective in complex combinatorial optimization problems such as scheduling and resource allocation [18]. In our formulation, the GA evolves a population of candidate schedules represented as vectors of start times, guided by a penalty-based cost function that reflects feasibility, energy consumption, and overall production time.

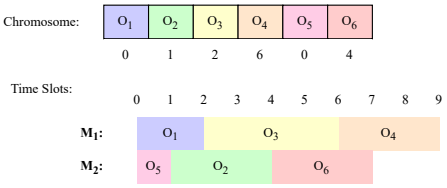

FIGURE 3: Chromosome representation and its equivalent schedule.

A. CHROMOSOME REPRESENTATION

Each individual in the population represents a candidate solution to the scheduling problem. A chromosome is encoded as an integer vector

$$X = [s_1, s_2, \ldots, s_n],$$

where each gene s_i denotes the starting time of operation i. The length of the chromosome corresponds to the total number of operations in the instance.

This representation provides a direct mapping between the vector and the temporal layout of the schedule. Machine conflicts, precedence violations, peak load, and makespan can all be derived directly from the vector of start times. Feasibility is not enforced directly at the encoding level; instead, violations incur penalties in the fitness function, which guides the GA toward valid schedules throughout evolution.

B. POPULATION INITIALIZATION

The initial population P_0 is generated using a job-based randomized heuristic. For each job, operations are assigned according to their prescribed order. The first operation of each job is assigned a random starting time, while subsequent operations begin immediately after the completion of their predecessor plus a random slack value. This method ensures that precedence constraints are always satisfied while promoting population diversity. Machine feasibility is not enforced at this stage and is handled through penalty terms in the cost function.

C. FITNESS EVALUATION

The fitness function used in our implementation corresponds to a weighted sum of penalties and performance indicators, formulated as

$$f(X) = \alpha P_{\text{mach}}(X) + \beta P_{\text{prec}}(X) + \gamma L_{\text{peak}}(X) + \delta C_{\max}(X), \tag{9}$$

where:

- $P_{\text{mach}}(X)$ counts the number of machine conflicts, i.e., overlapping operations assigned to the same machine,
- $P_{\text{prec}}(X)$ counts the violations of precedence constraints between consecutive operations of the same job,

- $L_{\text{peak}}(X)$ denotes the peak machine load, defined as the maximum number of simultaneously active machines over the entire horizon,
- $C_{\max}(X)$ corresponds to the makespan of the schedule.

The coefficients $(\alpha, \beta, \gamma, \delta)$ are selected to heavily penalize infeasible schedules, with α and β dominating the environmental and temporal terms. The GA performs direct minimization of (9), following DEAP's minimization-based fitness definition.

D. GENETIC OPERATORS

At each generation, new individuals are produced through selection, crossover, and mutation. Tournament selection with a small tournament size is used to balance exploration and selective pressure.

1) Crossover

A two-point crossover operator is employed. Two cut points are selected at random within the chromosome, and the segment between them is exchanged between the two parents. This operator preserves contiguous regions of the schedule while promoting genetic recombination.

2) Mutation

Mutation is performed using a multi-shift operator. Each gene s_i is mutated with probability p_m by adding an integer offset $\Delta \in [-k, k]$. The maximum shift magnitude k decreases linearly throughout the evolution, enabling broad exploration in early generations and fine-tuned exploitation in later ones.

A simple boundary clamping mechanism is applied immediately after mutation. If a start time shifts outside the valid scheduling horizon $[0, H]$, it is clamped to the nearest boundary:

$$s_i' = \max(0, \min(H, s_i + \Delta))$$

This ensures the chromosome always represents a valid set of start times relative to the horizon, even if the schedule itself remains infeasible regarding machine or precedence constraints.

E. REPLACEMENT AND ELITISM

A Generational Elitist replacement strategy is employed. At each generation, the N_{elite} best individuals from the current parent population are carried over directly to the next generation (elitism). The remaining $N - N_{\text{elite}}$ individuals are selected strictly from the newly generated offspring using tournament selection. This strategy ensures that the best solutions found so far are never lost, while the population is continuously refreshed with new genetic material derived from the offspring.

F. TERMINATION CRITERIA

The algorithm terminates after a fixed number of generations $G_{\max}$. No secondary stopping condition is used.

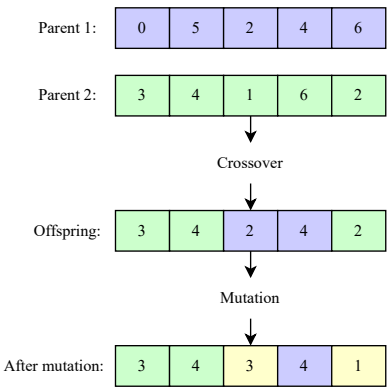

FIGURE 4: Two-point crossover and multi-shift mutation in the proposed GA.

G. PARAMETER CONFIGURATION

Due to the varying nature of each scheduling instance, the parameters used for the Genetic Algorithm are designed to scale with the complexity of the problem. Table 1 summarizes the configuration used for the standard **ft10** instance. Note that population size and the number of generations are dynamic, being calculated as $\max(15 \cdot |O|, 200)$ and $5 \cdot |O|$ respectively, where $|O|$ is the total number of operations in the instance.

TABLE 1: Genetic Algorithm parameters (ft10 instance)

Parameter	Value
Population size	$\approx$ 1500 (Dynamic)
Generations $G_{\max}$	$\approx$ 500 (Dynamic)
Crossover probability p_c	0.8
Mutation probability p_m	0.2
Tournament size	2
Elitism size	5
Penalty coefficients $(\alpha, \beta, \gamma, \delta)$	(1000, 1000, 10, 0.1)

Of course, these dynamic values won't be optimal for every single scheduling instance, thus, parameters such as population size or the number of generations may need to be manually tweaked in order to obtain a more optimized result.

H. ALGORITHM SUMMARY

The complete procedure is summarized in Algorithm 1. As feasibility is not enforced directly; instead, infeasible individuals naturally receive higher cost values and are therefore less likely to survive.

The proposed GA constitutes a simple, yet effective framework for energy-aware scheduling. By combining a start-time vector encoding with a penalty-based fitness function and a progressively refined mutation operator, the method is capable of guiding the search toward schedules with minimal peak load, low conflict levels, and acceptable makespan.

Algorithm 1 Genetic Algorithm for Scheduling via Start-Time Optimization

Initialize population P using randomized job-based initialization
Evaluate fitness of P
for $g \leftarrow 1$ **to** $G_{\max}$ **do**
    Update mutation shift magnitude based on progress $g/G_{\max}$
    Generate offspring O by applying crossover and mutation to P
    Evaluate fitness of offspring O
    $E \leftarrow$ Select best individuals from P (Elitism)
    $S \leftarrow$ Select remaining individuals from O (Tournament Selection)
    $P \leftarrow E \cup S$ ▷ Generational replacement
end for
return best individual found in P

V. RESULTS

In this section, we describe three algorithms developed to address the scheduling problem: a single-objective genetic algorithm, a multi-objective genetic algorithm, and a simulated annealing algorithm. The simulated annealing method serves as a baseline for evaluating the performance of the proposed genetic algorithms. All genetic-algorithm implementations were developed in Python 3 using the DEAP framework and executed in Google Colab [19]. For reproducibility, we fixed a random seed so that every execution yields identical results, ensuring that all algorithms are evaluated under the same conditions. However, parameter tuning was performed without a fixed seed to avoid any interaction between the random seed and the parameter selection process.

To test all algorithms, we use the problem instances FT06, FT10 and LA01.

A. SINGLE OBJECTIVE ALGORITHM

The parameters of the single-objective genetic algorithm are summarized in Table (2). The *Horizon* is defined as the sum of the durations of all operations, representing the maximum possible makespan when operations are executed sequentially. The population size is set to the greater value between 200 and fifteen times the number of operations, ensuring sufficient diversity even for small problem instances. The algorithm is executed for 200 generations, with crossover and mutation probabilities of 0.8 and 0.2, respectively. To preserve population diversity, the tournament size is fixed at 2, and the elitism mechanism retains the top 5 individuals.

The mutation shift parameter controls the magnitude of gene perturbation during mutation and is set to 0.7 times the Horizon.

Two parameters with significant influence on algorithm performance are the *creation start shift* and the *slack range*. Both affect the initialization of individuals. The creation start shift specifies the maximum allowable deviation of an operation's start time from zero during initialization, while the slack range determines the maximum offset allowed between consecutive jobs. The Creation start shift is set to 0.3 times the Horizon, and the slack range is 0.05 times the Horizon.

TABLE 2: Single Objective Genetic Algorithm parameters

Parameter	Value
HORIZON	$\sum d_{ops}$
POPULATION_SIZE	$\max(n_{\text{ops}} \cdot 15,\ 200)$
NUM_GENERATIONS	200
CROSS_PROBABILITY	0.8
MUT_PROBABILITY	0.2
TOURNAMENT_SIZE	2
ELITE_SIZE	5
MUTATION_START_SHIFT	$(HORIZON \cdot 0.7)$
END_SHIFT	1
CREATION_START_SHIFT	$(HORIZON \cdot 0.3)$
SLACK_RANGE	$(HORIZON \cdot 0.05)$

d_{ops} = duration of operations.
n_{ops} = number of operations.

Figures (5), (6), and (7) illustrate the evolution of the genetic algorithm across generations for each problem instance. The fitness curve decreases rapidly during the initial generations due to the penalty terms. Once the algorithm reaches zero conflicts, the convergence rate slows, and the curve stabilizes. Although the algorithm could theoretically be stopped after approximately 75 generations, we chose to extend the execution to allow further exploration of the search space and the possibility of finding improved solutions.

As shown in Figures (5), (6) and (7), even with a relatively high mutation probability and a multi-bit mutation operator, population diversity remains stable throughout most of the execution. This behavior is largely attributed to the slack mechanism, which prevents premature convergence. Diversity gradually decreases only during the final generations, when the population converges toward the best solutions.

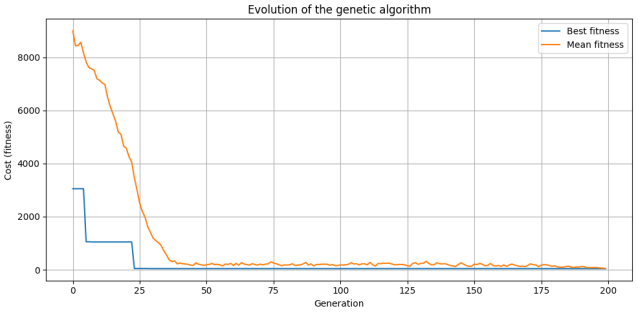

FIGURE 5: Evolution of the single objective genetic algorithm for the problem instance FT06.

B. MULTI OBJECTIVE GENETIC ALGORITHM

For this version of the genetic algorithm, several parameters were adjusted to improve the fitness results. The number of generations was increased to 500. The mutation start shift was set to 1.2 times the Horizon, while the creation start shift and the creation slack range were set to 0.1 and 0.2 times

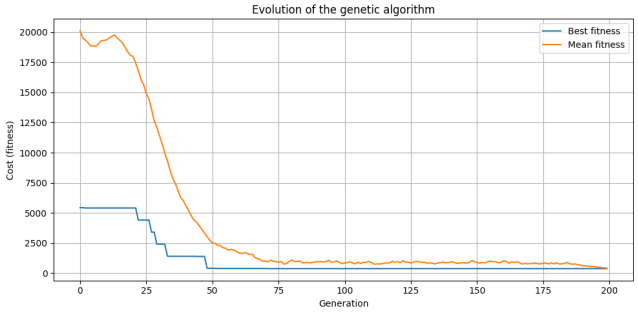

FIGURE 6: Evolution of the single objective genetic algorithm for the problem instance FT10.

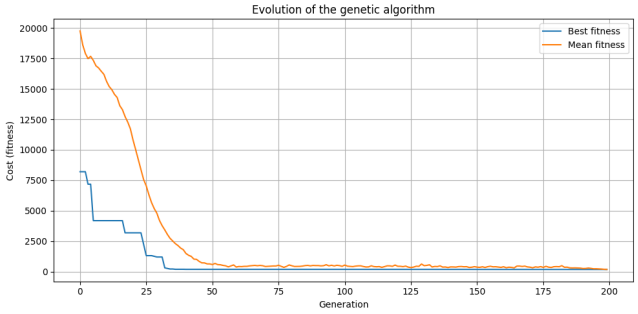

FIGURE 7: Evolution of the single objective genetic algorithm for the LA01 problem instance .

TABLE 3: Multi Objective Genetic Algorithm parameters

Parameter	Value
HORIZON	$\sum d_{ops}$
POPULATION_SIZE	$\max(n_{\text{ops}} \cdot 15,\ 200)$
NUM_GENERATIONS	500
CROSS_PROBABILITY	0.8
MUT_PROBABILITY	0.2
TOURNAMENT_SIZE	2
ELITE_SIZE	5
MUTATION_START_SHIFT	$(HORIZON \cdot 1.2)$
END_SHIFT	1
CREATION_START_SHIFT	$(HORIZON \cdot 0.1)$
SLACK_RANGE	$(HORIZON \cdot 0.2)$

d_{ops} = duration of operations.

n_{ops} = number of operations.

the Horizon, respectively. All other parameters remained unchanged. Table (3) provides a detailed description of the parameter configuration for this algorithm.

C. MULTI OBJECTIVE ALGORITHM

The fitness function now consists of two objectives. The first objective is defined as $\gamma \cdot L_{peak} + \delta \cdot C_{max}$, while the second objective is $\alpha \cdot P_{mach} + \beta \cdot P_{prec}$. The parameters α, β, δ and γ retain the same values as in the single-objective formulation (1000, 1000, 0.1, and 10, respectively). To support multi-objective optimization, the selection mechanism was replaced with NSGA-II, which is specifically designed for Pareto-based evolutionary selection.

Figures (8) and (10) show the feasible solutions on the Pareto front obtained by the algorithm for the *ft06* and *la01* instances. In contrast, Figure (9) presents the full Pareto front for the *ft10* instance, as no conflict-free solutions were found for this case. This result differs from the single-objective algorithm, which was able to produce a feasible solution for the same instance.

To identify the best individual on the Pareto front, we sorted the solutions by their penalty objective and selected the individual with the lowest corresponding fitness value.

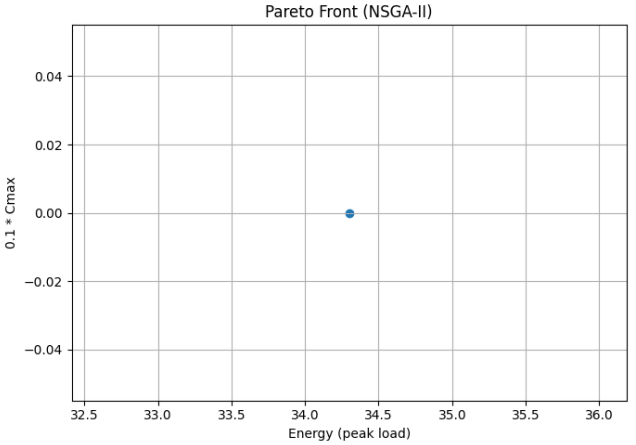

FIGURE 8: Feasible solutions in the pareto front of the multi objective genetic algorithm for the problem instance FT06.

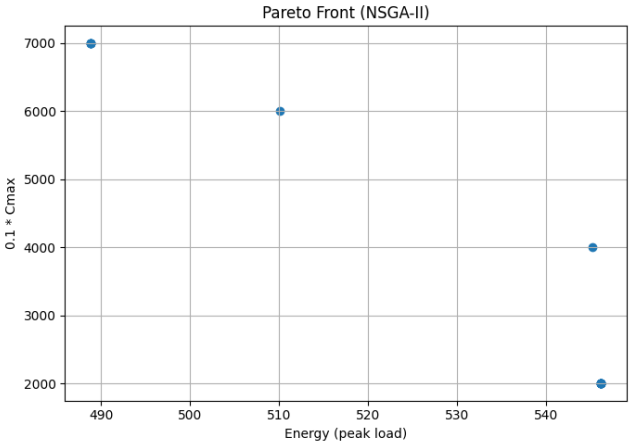

FIGURE 9: Pareto front of the multi objective genetic algorithm for the problem instance FT10.

D. SIMULATED ANNEALING ALGORITHM

The simulated annealing algorithm is based on the analogy between the simulation of the annealing of solids and the problem of solving large combinatorial optimization problems [20]. The algorithm explores the solution space by iteratively applying small perturbations to the current solution.

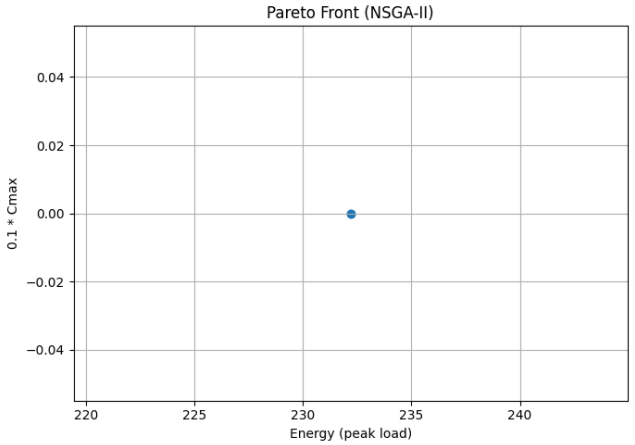

FIGURE 10: Feasible solutions in the pareto front of the multi objective genetic algorithm for the problem instance LA01.

TABLE 4: Simulated Annealing Algorithm parameters

Parameter	Value
MAX_ITERS	10,000,000
T_INIT	2500
T_MIN	$1e-3$
ALPHA	0.9999
MAX_SHIFT	3

At each iteration, a new candidate solution is accepted if it improves the objective function; otherwise, it may still be accepted with a probability that decreases as the algorithm's temperature parameter is reduced. This mechanism allows the method to escape local optima during the early stages and gradually converge toward high-quality solutions as the temperature approaches zero.

Table (4) provides a detailed description of the parameters used in the simulated annealing algorithm. These parameters include the total number of iterations, the initial and minimum temperatures, the regularization parameter α that controls the cooling schedule, and the maximum shift applied when generating a neighboring solution.

For the fitness evaluation, the simulated annealing algorithm uses the same objective function as the single-objective genetic algorithm. The neighborhood function generates a new candidate solution by applying a random number of modifications, where

$$n \in \left(0, \left\lfloor \frac{\text{length(individual)}}{10} \right\rfloor\right).$$

For each modification, a random element of the solution is selected and shifted by a random value

$$k \in (-\text{max\_shift}, \text{max\_shift}).$$

.

If the resulting value exceeds the Horizon, it is clipped to the Horizon to ensure a valid solution.

Figures (11), (12), and (13) show the evolution of the simulated annealing algorithm across iterations for each problem instance. As observed, all instances exhibit substantial variability during the initial iterations, which encourages exploration of the solution space. However, toward the end of the process, the curves stabilize as the algorithm converges, favoring the identification of high-quality solutions.

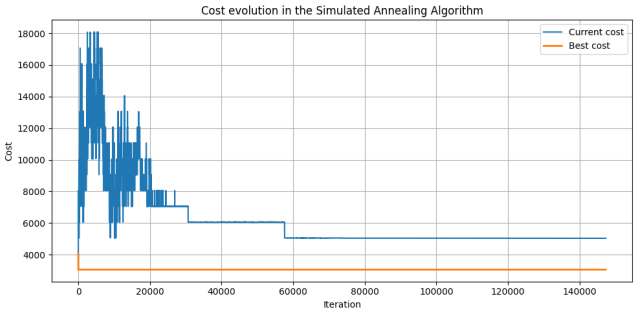

FIGURE 11: Evolution of the simulated annealing algorithm for the problem instance FT06.

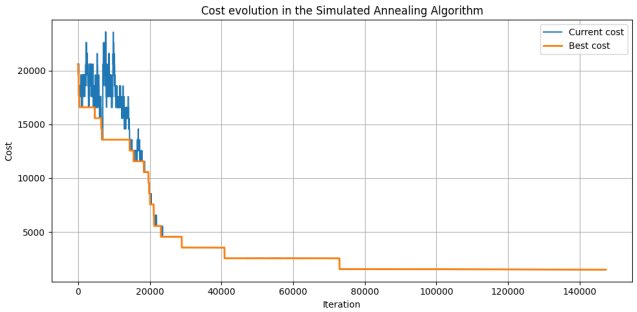

FIGURE 12: Evolution of the simulated annealing algorithm for the problem instance FT10.

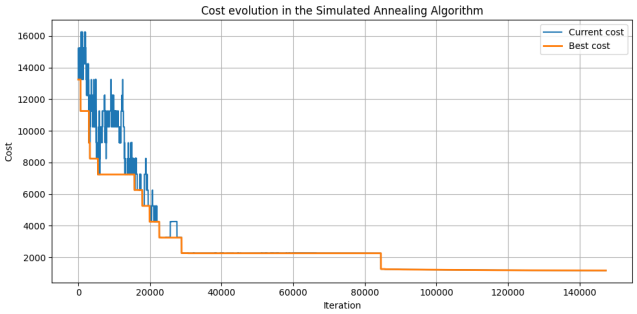

FIGURE 13: Evolution of the simulated annealing algorithm for the problem instance LA01.

In Figure (11), the variation observed in the algorithm's behavior is greater than in the other problem instances, and its convergence is notably poorer compared to the genetic algorithms. A plausible explanation is that the *ft06* instance is

relatively simple, causing the selected parameters to induce excessive exploration. As a result, instead of helping the algorithm escape local optima, the exploration phase may lead it to settle prematurely into one.

E. ALGORITHM COMPARISON

Table (5) summarizes the best fitness values obtained by all algorithms for each problem instance. As shown in the table, the single-objective genetic algorithm is the only method that achieves conflict-free solutions in all experiments. It also yields the best fitness values for all instances except *ft06*, where the multi-objective algorithm performs slightly better. This difference may be attributed to an insufficient number of generations or a less effective initial exploration in the single-objective configuration.

The multi-objective algorithm successfully resolves conflicts for the *ft06* and *la01* instances, but fails to produce a conflict-free solution for *ft10*. These results suggest that the multi-objective approach is more effective for small- to medium-sized problem instances.

Finally, the simulated annealing algorithm does not achieve conflict-free solutions for any instance. Nevertheless, it exhibits comparatively better performance on medium and large instances, outperforming the multi-objective genetic algorithm in the *ft10* case.

VI. CONCLUSION

From this investigation, several conclusions can be drawn.

- Among the algorithms evaluated, the single-objective genetic algorithm demonstrates the strongest overall performance. It consistently produces feasible solutions across all tested instances and achieves better fitness values than the multi-objective approach for medium- and large-sized problems.
- For problem instances with fewer operations and/or machines, the multi-objective genetic algorithm remains a viable alternative. Its ability to explore trade-offs between objectives may provide advantages in scenarios where solution diversity is desirable.
- In general, genetic algorithms outperform the simulated annealing method. However, simulated annealing still attains competitive results for medium- and large-sized problems.
- The proposed approach successfully identifies feasible schedules capable of reducing production time in flow-line environments. This methodology has the potential not only to optimize throughput, but also to contribute to energy-efficient manufacturing by minimizing idle times and operational overlap.

Although the multi-objective genetic algorithm underperformed compared with the single-objective version in most cases, it shows clear potential for improvement. Future work will focus on improving this approach by incorporating adaptive parameter control, improved initialization strategies, and hybridization with local search methods. In addition, we plan to investigate the scalability to larger industrial instances and evaluate the algorithms under real-world manufacturing constraints such as machine availability, maintenance windows, and energy cost variability.

TABLE 5: Experimentation results algorithm comparison

Algorithm	FT06		FT10		LA01	
	Results	Conflicts	Results	Conflicts	Results	Conflicts
Single Objective GA	41.90	0	376.00	0	175.90	0
Multi Objective GA	(34.3, 0.0)	0	(546.2, 2000.0)	2	(232.20000000000002, 0.0)	0
Simulated Annealing	3047.40	3	1499.80	1	1171.40	1

.

ACKNOWLEDGMENT

We would like to thank Loyola University and our instructors, Daniel Gutiérrez Reina and Alejandro Tapia Córdoba, for providing the knowledge and guidance necessary to conduct this research.

APPENDIX A CODE

To explore the research in greater depth, you can access the code in [19].

REFERENCES

[1] Ahmed Kouider and Hacène Ait haddadène. A bi-objective branch-and-bound algorithm for the unit-time job shop scheduling : A mixed graph coloring approach. Computers & Operations Research, 132:105319, August 2021.
[2] Candice Destouet, Houda Tlahig, Belgacem Bettayeb, and Bélahcène Mazari. Flexible job shop scheduling problem under Industry 5.0: A survey on human reintegration, environmental consideration and resilience improvement. Journal of Manufacturing Systems, 67:155–173, April 2023.
[3] Congressional Budget Office. Emissions of Greenhouse Gases in the Manufacturing Sector | Congressional Budget Office, February 2024.
[4] Janak Bhatta. File:Air pollution3.jpg - Wikimedia Commons.
[5] Tiny Wijerathna Ekanayake, Pavani Subasinghe, Shawn Ragel, Anjalie Gamage, and Suchini Attanayaka. Intelligent Timetable Scheduler: A Comparison of Genetic, Graph Coloring, Heuristic and Iterated Local Search Algorithms. In 2019 International Conference on Advancements in Computing (ICAC), pages 85–90, December 2019.
[6] Suman De and Vinod Vijayakumaran. An Efficient Algorithm in Project Management for Resource Scheduling and Conflict Management using Graph Coloring Technique. In 2020 IEEE International Conference for Innovation in Technology (INOCON), pages 1–6, November 2020.
[7] Suman De. An efficient technique of resource scheduling in cloud using graph coloring algorithm. Global Transitions Proceedings, 3(1):169–176, June 2022.
[8] Olivier Goudet, Béatrice Duval, and Jin-Kao Hao. Population-based gradient descent weight learning for graph coloring problems. Knowledge-Based Systems, 212:106581, January 2021.
[9] Raja Marappan and Gopalakrishnan Sethumadhavan. Complexity Analysis and Stochastic Convergence of Some Well-known Evolutionary Operators for Solving Graph Coloring Problem. Mathematics, 8(3):303, March 2020. Publisher: Multidisciplinary Digital Publishing Institute.
[10] Jocelin Cailloux, Nicolas Zufferey, and Olivier Gallay. Graph coloring approaches for a production planning problem with makespan and setup penalties in a product-wheel context. Discrete Applied Mathematics, 355:200–222, October 2024.
[11] Yuri N. Sotskov and Evangelina I. Mihova. Scheduling Multiprocessor Tasks with Equal Processing Times as a Mixed Graph Coloring Problem. Algorithms, 14(8):246, August 2021. Publisher: Multidisciplinary Digital Publishing Institute.
[12] Yuri N. Sotskov. Mixed Graph Colorings: A Historical Review. Mathematics, 8(3):385, March 2020. Publisher: Multidisciplinary Digital Publishing Institute.
[13] Ming Gao and Jing Hu. Graph Coloring Algorithm Based on Minimal Cost Graph Neural Network. IEEE Access, 12:168000–168009, 2024.
[14] Zhongwei Zhang, Renzhong Tang, Tao Peng, Liyan Tao, and Shun Jia. A method for minimizing the energy consumption of machining system: integration of process planning and scheduling. Journal of Cleaner Production, 137:1647–1662, 2016.
[15] Tamy0612. Jsplib. https://github.com/tamy0612/JSPLIB, 2022. Accessed: [08 november 2025].
[16] David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.
[17] John H. Holland. Adaptation in Natural and Artificial Systems. MIT Press, 1992.
[18] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. Handbook of Evolutionary Computation. IOP Publishing and Oxford University Press, 1997.
[19] Lucía García Lado Alejandro Sosa Corral. Graph coloring models for production line scheduling optimization. https://github.com/Lucia-Garcia-Lado/Graph-Coloring-Models-for-Production-Line-Scheduling-Optimization.git, 2025.
[20] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In Simulated annealing: Theory and applications, pages 7–15. Springer, 1987.

FIRST A. AUTHOR (M'76–SM'81–F'87) and all authors may include biographies. Biographies are often not included in conference-related papers. This author became a Member (M) of IEEE in 1976, a Senior Member (SM) in 1981, and a Fellow (F) in 1987. The first paragraph may contain a place and/or date of birth (list place, then date). Next, the author's educational background is listed. The degrees should be listed with type of degree in what field, which institution, city, state, and country, and year the degree was earned. The author's major field of study should be lower-cased.

The second paragraph uses the pronoun of the person (he or she) and not the author's last name. It lists military and work experience, including summer and fellowship jobs. Job titles are capitalized. The current job must have a location; previous positions may be listed without one. Information concerning previous publications may be included. Try not to list more than three books or published articles. The format for listing publishers of a book within the biography is: title of book (publisher name, year) similar to a reference. Current and previous research interests end the paragraph. The third paragraph begins with the author's title and last name (e.g., Dr. Smith, Prof. Jones, Mr. Kajor, Ms. Hunter). List any memberships in professional societies other than the IEEE. Finally, list any awards and work for IEEE committees and publications. If a photograph is provided, it should be of good quality, and professional-looking. Following are two examples of an author's biography.

LUCÍA GARCÍA LADO was born in Burgos, Spain in 2003. He received the degree in Computer Engineering from the University of Burgos, Spain, in 2025, with a specialization in Computing and Information Systems.

From July 2024 to August 2024, she was an intern at Antolín, where she worked on computer vision and labeling automation. From September 2024 to June 2025, she was PTGAS (Technical Staff for Management, Administration, and Services) at the University of Burgos, associated with the OptiDit research project, where she worked on developing Artificial Intelligence models for time series prediction to forecast the power generated by wind farm turbines.

She is the co-author of 1 article in the WWWE 2024 (VI Workshop on Wind and Waves Energy). Her research interests include applications of Artificial Inteligence to optimze energy consumption.

In 2018 she received the third prize in the "This Is My Invention" contest organized by the CSIC (Spanish National Research Council).

THIRD C. AUTHOR, JR. (M'87) received the B.S. degree in mechanical engineering from National Chung Cheng University, Chiayi, Taiwan, in 2004 and the M.S. degree in mechanical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2006. He is currently pursuing the Ph.D. degree in mechanical engineering at Texas A&M University, College Station, TX, USA.

From 2008 to 2009, he was a Research Assistant with the Institute of Physics, Academia Sinica, Tapei, Taiwan. His research interest includes the development of surface processing and biological/medical treatment techniques using nonthermal atmospheric pressure plasmas, fundamental study of plasma sources, and fabrication of micro- or nanostructured surfaces.

Mr. Author's awards and honors include the Frew Fellowship (Australian Academy of Science), the I. I. Rabi Prize (APS), the European Frequency and Time Forum Award, the Carl Zeiss Research Award, the William F. Meggers Award and the Adolph Lomb Medal (OSA).