Date of publication xxxx 00, 0000, date of current version December 14, 2025.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Energy-Aware Production Line Scheduling Optimization using Graph Coloring and Genetic Algorithms

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. Both genetic algorithms demonstrate superior performance compared to the simulated annealing baseline, with a 89.91% average difference in overall fitness of the solutions between the approaches. 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, as highlighted by [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, as discussed by [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 of manufacturing has become a critical concern, particularly due to the high energy demands of production systems. For instance, according to data reported by the U.S. Congressional Research Service [3], production lines account for approximately 12% of total emissions in the United States. due to high energy consumption and inefficient resource allocation. As global industries expand, these inefficiencies compound, amplifying these environmental burdens.

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.

As reported in [4], [5], and [6], graph coloring problem–based algorithms have been successfully applied to scheduling tasks. Accordingly, we will cover a graph-theoretic optimization method based on the graph coloring problem to reduce idle times, balance workloads, and minimize energy consumption, ultimately enhancing both operational and environmental performance.

In summary, the main contributions of this paper are:

- We formulate an energy-aware production scheduling problem as a graph-coloring model, explicitly incorporating environmental considerations into job-shop scheduling.
- We develop a single-objective and a multi-objective evolutionary algorithm based on Genetic Algorithms and compare them with a Simulated Annealing (SA) baseline.
- We evaluate the proposed approach on standard JSPLIB (Job Shop Scheduling Problem Library) benchmarks (FT06, FT10, LA01), analyzing feasibility, convergence, and environmental performance.

To the best of our knowledge, no previous work has combined a pure graph coloring formulation with genetic algorithms specifically for peak-load minimization in this specific job-shop context.

The remainder of the paper is structured as follows. Section II reviews the existing literature on graph coloring and its applications to scheduling problems. Section III formally introduces the energy-aware Job Shop Scheduling problem and presents its formulation as a graph coloring task, including objective functions, constraints, and assumptions. Section IV describes the proposed evolutionary framework, detailing the chromosome representation, initialization strategy, fitness function, genetic operators, and the configuration of both the single-objective and multi-objective algorithms. Section V reports the experimental evaluation, including the performance of the genetic algorithms and the simulated annealing baseline on benchmark instances from JSPLIB. Finally, Section VI summarizes the main findings and discusses potential directions for future research.

II. RELATED WORK

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 [4] 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, [7] and [8] 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 [5], 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. [6] 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 [9] 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 [10]. 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 [11], 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. [12] 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. Going a step further, recent surveys such as [13], [14], and [15] provide comprehensive reviews of energy-efficient scheduling problems, identifying evolutionary approaches—particularly genetic algorithms—as prominent and effective solution techniques.

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 [9]. Figure 1 shows an example of a solved graph coloring problem.

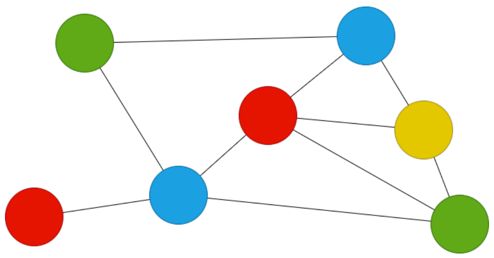


FIGURE 1: Example of a solved graph coloring problem G(7,10).

The Graph Scheduling Problem has been found to be equivalent to General Shop Scheduling Problems [10]. 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 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. [10]

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

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 [16]. 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, although, unlike classical graph coloring, constraints are not enforced strictly; instead, they will be evaluated through integer penalties.

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 stated earlier, 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:

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

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,
- P_{mach} and P_{prec} represent the count of machine and precedence conflicts, respectively,
- γ and δ are penalty parameters.

D. MULTI-OBJECTIVE PROBLEM DEFINITION

Similarly to the single-objective formulation, the multi-objective algorithm is also constrained by (3), (5), and (7). In this case, however, the overall optimization goal is decomposed into two complementary objectives: scheduling efficiency and conflict minimization. Based on this decomposition, the multi-objective problem can be formally expressed as follows:

$$\begin{aligned} \min & \begin{cases} f_1(x) = \gamma\, L_{\text{peak}}(x) + \delta\, C_{\max}(x), \\ f_2(x) = \alpha\, P_{\text{mach}}(x) + \beta\, P_{\text{prec}}(x), \end{cases} \\ \text{s.t. } & P_{\text{mach}}(x) = 0, \\ & P_{\text{prec}}(x) = 0, \\ & C_{\max}(x) > 0. \end{aligned} \tag{9}$$

where

- $L_{\text{peak}}(x)$ is the peak load,
- $C_{\max}(x)$ is the makespan,
- $P_{\text{mach}}(x)$ and $P_{\text{prec}}(x)$ represent the number of machine and precedence conflicts, respectively,
- α, β, γ and δ are penalty parameters.

E. ASSUMPTIONS

1) **Assumption 1:** In this problem, we assume that reducing energy consumption leads to a corresponding decrease in environmental pollution.
2) **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.
3) **Assumption 3:** We assume that every operation takes an exact number of time slots.
4) **Assumption 4:** We assume time slots of 1 minute.
5) **Assumption 5:** We assume that all machines are available at any time and remain in working condition.
6) **Assumption 6:** We assume that no external factors alter the original scheduling problem.

We make these assumptions that simplify the scheduling instances and operations so that we can focus on the scheduling logic and proposed algorithm on this paper. Future work will take these elements into account to further refine the algorithm and make it even more applicable in more realistic scenarios.

F. DATA SET ACQUISITION

The data set used in this study was obtained from [17]. 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 FT06 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 [18], [19]. They have been shown to be effective in complex combinatorial optimization problems such as scheduling and resource allocation [20]. 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.

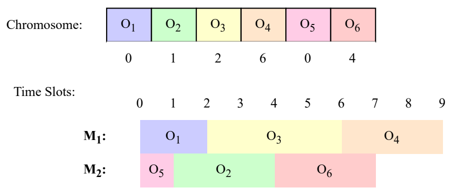


FIGURE 2: Chromosome representation and its equivalent schedule.

A. CHROMOSOME REPRESENTATION

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

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

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, which can be observed in Figure 2 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 reduces the number of precedence violations in the initial population. Machine feasibility is not enforced at this stage and is handled through penalty terms in the cost function. Although more sophisticated heuristics such as shifting-bottleneck or dispatching rules could be incorporated into the initialization phase, we intentionally adopt a minimalist approach to isolate the contribution of the evolutionary search itself.

C. FITNESS EVALUATION

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

$$f(X) = \alpha P_{\text{mach}}(X) + \beta P_{\text{prec}}(X) + \gamma L_{\text{peak}}(X) + \delta C_{\max}(X), \tag{10}$$

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 (10), 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. A visual example of the crossover and mutation processes can be observed in Figure 3

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.

We set the elite size to 5 for both genetic algorithms. This value was selected to ensure that a small yet representative subset of high-quality solutions is preserved across

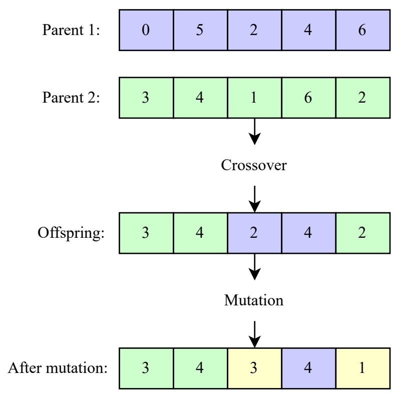


FIGURE 3: Two-point crossover and multi-shift mutation in the proposed GA.

generations, regardless of the dynamic population size. Using a fixed and relatively small elite size prevents excessive selection pressure as the population grows, while still guaranteeing that the best individuals are not lost. Moreover, since tournament selection already introduces a degree of elitism, limiting the number of explicitly preserved individuals helps maintain population diversity and reduces the risk of premature convergence. Empirically, this configuration provided a good balance between solution preservation and exploration across problem instances of varying sizes.

F. TERMINATION CRITERIA

The algorithm terminates after a fixed number of generations $G_{\max}$. More advanced termination criteria (such as convergence thresholds or lack-of-improvement criteria) were not used to ensure consistent and comparable runtimes across all benchmark instances.

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 number of generations is dynamic, being calculated as $\max(15 \cdot |O|, 200)$ and $\max(5 \cdot |O|, 200)$ respectively, where $|O|$ is the total number of operations in the instance.

Naturally, these dynamic values will not 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.

TABLE 1: Genetic Algorithm parameters (ft10 instance example)

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)

Algorithm 1 Genetic Algorithm for Scheduling via Start-Time Optimization

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

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.

I. SIMULATED ANNEALING ALGORITHM

To evaluate the performance of the proposed genetic algorithms, we employ a simulated annealing (SA) algorithm as a baseline. SA is a well-established metaheuristic commonly used for combinatorial optimization problems and provides a suitable reference for assessing solution quality and convergence behavior.

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 [21]. The algorithm explores the solution space by iteratively applying small perturbations to the current solution.

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.

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, we select a random element of the solution and shift it 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.

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 [22] and executed in Google Colab [23] with the following runtime settings:

- Runtime Type: Python 3
- Hardware Accelerator: CPU
    1) System Ram: 12.7 GB
    2) Disk: 107.7 GB
- Runtime Version: Latest (recommended)

For reproducibility, we fixed a set of seeds so that every execution yields identical results, ensuring that all algorithms are evaluated under the same conditions. However, we performed parameter tuning without fixed seeds 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.

All parameters in each algorithm were selected using a trial-and-error approach, with two to three executions performed to ensure that changes in parameter values had a significant impact on the algorithm's results and that the observed improvements were not due to the algorithm's inherent randomness.

A. SINGLE OBJECTIVE ALGORITHM

The parameters of the single-objective genetic algorithm are summarized in Table 2. We define the *Horizon* as the sum of the durations of all operations, representing the maximum possible makespan when operations are executed sequentially. We set the population size to the greater value between 200 and fifteen times the number of operations, ensuring sufficient diversity even for small problem instances. We execute the algorithm for a minimum of 200 generations, with crossover and mutation probabilities of 0.8 and 0.2, respectively. To preserve population diversity, we fix the tournament size 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. We define the creation start shift as 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. We set the creation start shift to 0.3 times the Horizon, and the slack range to 0.05 times the Horizon.

TABLE 2: Single Objective Genetic Algorithm parameters

Parameter	Value
HORIZON	$\sum d_{ops}$
Population Size	$\max(n_{\text{ops}} \cdot 15,\ 200)$
Number of Generations	$\max(n_{\text{ops}} \cdot 5,\ 200)$
Crossover Probability	0.8
Mutation 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 4, 5, and 6 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 4, 5 and 6, 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.

Even though problem instances FT10 and LA01 are more complex, Figures 5 and 6 show a fast convergence behavior very similar to that observed for the FT06 instance in Figure 4. In all cases, conflict-free solutions are reached as the mean fitness approaches zero-conflict values, which can be attributed to the applied elitism strategy. While this elitism promotes rapid convergence, it also entails that once a feasible solution is found, further fitness improvements become progressively slower and may nearly stagnate. However, as will be shown in Tables 5, 6, and 7, the solutions obtained are sufficiently efficient when compared to the other algorithms.

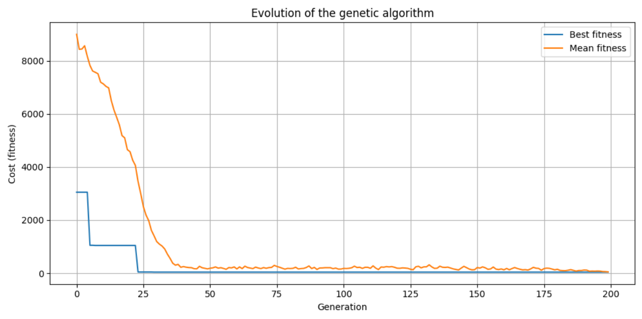


FIGURE 4: Evolution of the single objective genetic algorithm for the problem instance FT06.

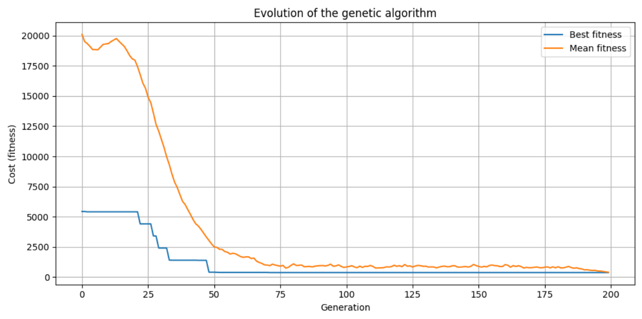


FIGURE 5: Evolution of the single objective genetic algorithm for the problem instance FT10.

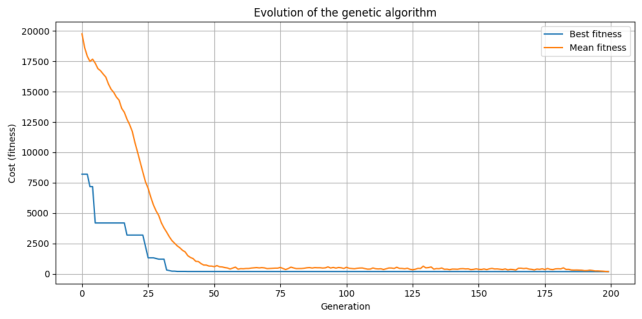


FIGURE 6: Evolution of the single objective genetic algorithm for the LA01 problem instance .

B. MULTI-OBJECTIVE GENETIC ALGORITHM

For this version of the genetic algorithm, several parameters were adjusted to improve the fitness results. We increased the minimum number of generations to 500. We set the mutation start shift to 1.2 times the Horizon, while we set the creation start shift and the creation slack were set to 0.1 and 0.2 times the Horizon, respectively. All other parameters remained unchanged. Table 3 provides a detailed description of the parameter configuration for this algorithm.

TABLE 3: Multi-Objective Genetic Algorithm parameters

Parameter	Value
HORIZON	$\sum d_{ops}$
Population Size	$\max(n_{\text{ops}} \cdot 15,\, 200)$
Number of Generations	$\max(n_{\text{ops}} \cdot 5,\, 500)$
Crossover Probability	0.8
Mutation 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.

As it can be observed, the "Number of Generations" parameter differs from the single-objective algorithm $(\max(n_{\text{ops}} \cdot 5,\, 500)$ instead of $\max(n_{\text{ops}} \cdot 5,\, 200))$ This is due to the differences between the two algorithms, requiring a higher number of minimum generations.

The selected configuration was determined based on our experimentation aimed at identifying parameter values that provide robust performance within a reasonable execution time. As stated before, they could be fine tuned to a greater extent with further experimentation, but from all our testing these values seem to offer a good balance between results and execution time.

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, we replaced the selection mechanism with NSGA-II [24], which is specifically designed for Pareto-based evolutionary selection.

Figures 7 and 9 show the feasible solutions on the Pareto front obtained by the algorithm for the *FT06* and *LA01* instances. In contrast, Figure 8 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.

Although Figures 7 and 9 display only a single point, multiple feasible solutions were obtained in both cases. However, all these solutions share identical objective values, which results in a single visible point on the Pareto front. This behavior is likely a consequence of the elitism and selection mechanisms employed by the algorithm, which tend to promote rapid convergence toward a small set of dominant solutions, thereby limiting population diversity.

Figure 8 provides a broader view of the Pareto front for the FT10 instance, where all solutions are infeasible.

The Figure illustrates how the algorithm explores different trade-offs among infeasible solutions, with some individuals prioritizing reductions in peak load and others focusing on minimizing the number of conflicts. Nevertheless, the Pareto front remains relatively narrow, indicating limited diversity among the solutions identified for this instance.

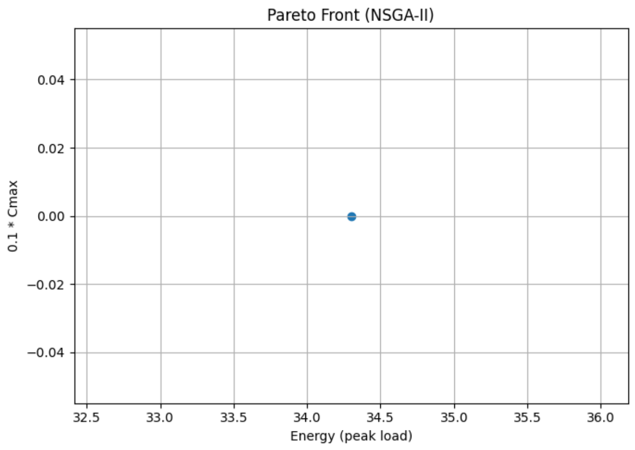


FIGURE 7: Feasible solutions in the Pareto front of the multi-objective genetic algorithm for the problem instance FT06.

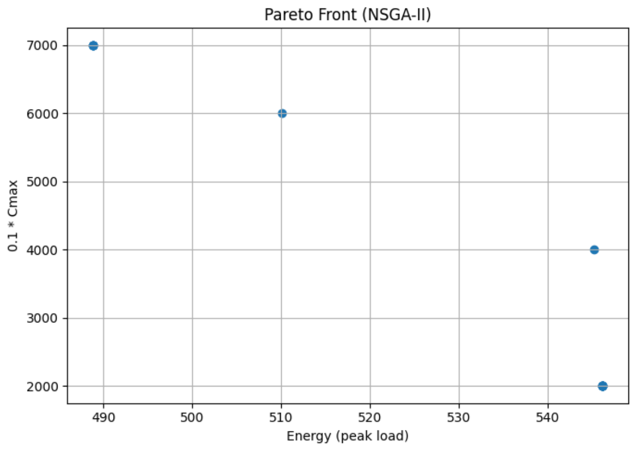


FIGURE 8: Pareto front of the multi-objective genetic algorithm for the problem instance FT10.

C. SIMULATED ANNEALING ALGORITHM

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.

Figures 10, 11, and 12 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

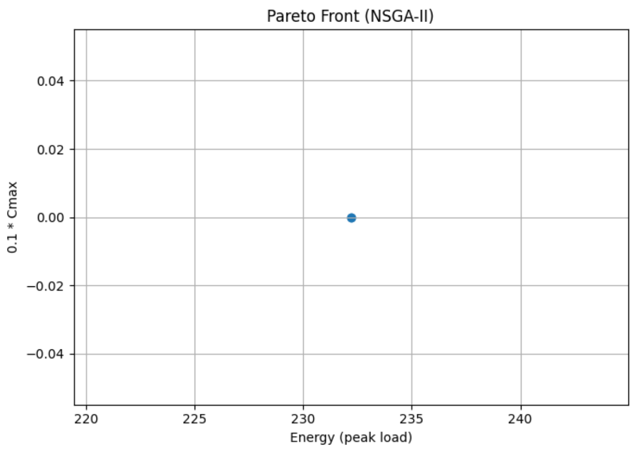


FIGURE 9: Feasible solutions in the Pareto front of the multi-objective genetic algorithm for the problem instance LA01.

TABLE 4: Simulated Annealing Algorithm parameters

Parameter	Value
Maximum Iterations	10,000,000
Initial Temperature	2500
Minimal Temperature	$1e-3$
α	0.9999
Maximum Shift	3

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.

In Figure 10, 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.

Contrary to the behavior observed in Figure 10, Figures 11 and 12 exhibit a clearer convergence pattern once the exploration phase concludes, with a consistent downward trend in the fitness values across iterations. This behavior is consistent with the expected dynamics of the simulated annealing algorithm and indicates that, for these more complex instances, the balance between exploration and exploitation is more effective. In these cases, the initial exploration allows the algorithm to escape poor-quality regions of the search space, while the subsequent exploitation phase progressively refines the solutions toward higher-quality configurations.

D. ALGORITHM COMPARISON

Tables 5, 6, and 7 summarize the best fitness values obtained by all algorithms for each problem instance, reporting both the mean and standard deviation of the fitness and execution time across all runs. The algorithm was executed 10 times for the *FT06* and *LA01* instances and 5 times for the *FT10* instance, as due to it's larger size, it took a lot longer to

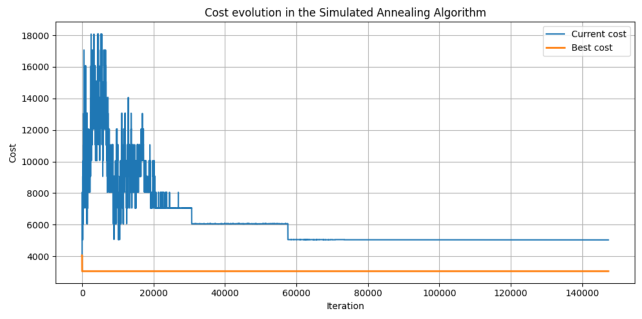


FIGURE 10: Evolution of the simulated annealing algorithm for the problem instance FT06.

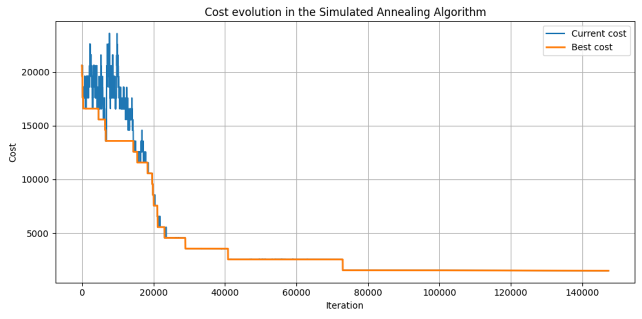


FIGURE 11: Evolution of the simulated annealing algorithm for the problem instance FT10.

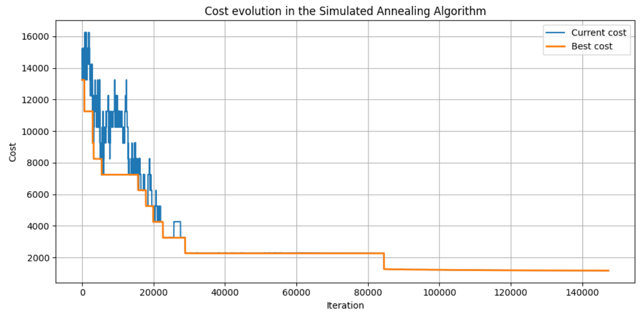


FIGURE 12: Evolution of the simulated annealing algorithm for the problem instance LA01.

TABLE 5: Results for FT06 Instance (Mean $\pm$ Standard Deviation, execution time (seconds), $N = 10$)

Algorithm	Cost / Obj	Conflicts	Execution Time
Single Objective GA	42.34 ± 2.62	0 ± 0	10.24 ± 1.77
Multi-Objective GA	36.32 ± 0.61	0.0 ± 0	140.21 ± 6.19
Simulated Annealing	3139.26 ± 696.79	3.0 ± 0	7.83 ± 0.58

TABLE 6: Results for FT10 Instance (Mean $\pm$ Standard Deviation, execution time (seconds), $N = 5$)

Algorithm	Cost / Obj	Conflicts	Execution Time
Single Objective GA	461.22 ± 307.97	0 ± 0	223.4 ± 2.18
Multi-Objective GA	2535.76 ± 912.78	2.0 ± 0.89	1796.95 ± 14.25
Simulated Annealing	3923.22 ± 2301.74	3.0 ± 2.0	30.14 ± 0.53

TABLE 7: Results for LA01 Instance (Mean $\pm$ Standard Deviation, execution time (seconds), $N = 10$)

Algorithm	Cost / Obj	Conflicts)	Execution Time
Single Objective GA	188.11 ± 6.78	0 ± 0	29.95 ± 1.74
Multi-Objective GA	424.98 ± 408.94	0.2 ± 0.4	447.10 ± 2.98
Simulated Annealing	581.32 ± 489.95	0 ± 0	15.05 ± 0.36

execute (as can be observed in each table's execution time). As shown in the table, the genetic algorithms perform way better than the Simulated Annealing baseline, with an average cost reduction of 89.91% between the three instances (comparing SA with the better performing GA in each instance). Among them, 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 only achieves conflict-free solutions for *LA01*. Nevertheless, it exhibits comparatively better performance on medium and large instances, outperforming the multi-objective genetic algorithm in that case.

Regarding execution time, as expected, simulated annealing is the fastest algorithm, since the genetic algorithms must evaluate the fitness function for a large number of individuals at each generation. Nevertheless, the single-objective genetic algorithm exhibits competitive execution times for small- and medium-sized problem instances. For larger instances, however, its execution time increases significantly; for example, as shown in Table 6, it requires 223.4 ± 2.18 seconds (nearly four minutes) to complete. Depending on production requirements, such execution times may be impractical. However, as previously discussed in the analysis of Figures 4, 5, and 6, the algorithm could be terminated earlier—once a feasible solution is found—at the cost of a slightly worse objective value but with a substantial reduction in execution time, still yielding more robust solutions than simulated annealing.

The multi-objective genetic algorithm presents the highest execution times across all problem instances. When combined with its inability to consistently find conflict-free solutions—particularly for larger instances—this significantly limits its applicability in real-world scenarios. Nevertheless, in contexts where execution time is less critical and solution quality for small problem instances is the primary concern (e.g., offline scheduling or next-day production planning), the multi-objective approach may still represent a viable alternative.

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. In addition, it exhibits a competitive execution time, making it suitable for practical and time-constrained scheduling scenarios.
- 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; however, its higher execution time must be carefully considered when assessing its applicability in time-sensitive environments.
- 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. We will focus on improving this approach in future work. In addition, we plan to investigate the scalability to larger, real industrial instances, rather than synthetic instances, and we will evaluate the algorithms under real-world manufacturing constraints such as machine availability, maintenance windows, and energy cost variability.

.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

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 [23].

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] 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.

[5] 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.

[6] Suman De. An efficient technique of resource scheduling in cloud using graph coloring algorithm. Global Transitions Proceedings, 3(1):169–176, June 2022.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Yuri N. Sotskov. Mixed Graph Colorings: A Historical Review. Mathematics, 8(3):385, March 2020. Publisher: Multidisciplinary Digital Publishing Institute.

[12] Ming Gao and Jing Hu. Graph Coloring Algorithm Based on Minimal Cost Graph Neural Network. IEEE Access, 12:168000–168009, 2024.

[13] João M. R. C. Fernandes, Seyed Mahdi Homayouni, and Dalila B. M. M. Fontes. Energy-efficient scheduling in job shop manufacturing systems: A literature review. Sustainability, 14(10), 2022.

[14] Ahmed Missaoui, Cemalettin Ozturk, Barry O'Sullivan, and Michele Garraffa. Energy efficient manufacturing scheduling: A systematic literature review, 2023.

[15] Kaizhou Gao, Yun Huang, Ali Sadollah, and Ling Wang. A review of energy-efficient scheduling in intelligent production systems. Complex & Intelligent Systems, 6(2):237–249, July 2020.

[16] 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.

[17] Tamy0612. Jsplib. https://github.com/tamy0612/JSPLIB, 2022. Accessed: [08 november 2025].

[18] David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

[19] John H. Holland. Adaptation in Natural and Artificial Systems. MIT Press, 1992.

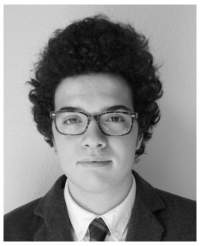
[20] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. Handbook of Evolutionary Computation. IOP Publishing and Oxford University Press, 1997.

[21] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In Simulated annealing: Theory and applications, pages 7–15. Springer, 1987.

[22] F.-A. Fortin, F.-M. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. Deap: Distributed evolutionary algorithms in python. GitHub: https://github.com/DEAP/deap, 2012. Accessed: 30 November 2025.

[23] 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.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.



ALEJANDRO SOSA CORRAL is a University of Sevilla graduate born in 2002, receiving his degree in Software Engineering in 2025 after returning from an Erasmus exchange in Tromsø, Norway, where he resided at The Arctic University of Norway as an exchange student from August 2022 to June 2023.

After finishing his TFG, he worked as an intern at Insinno Spain from July 2024 to January 2025, where he worked as a full-stack Web Application Developer for the duration of the internship, working with various development frameworks in different programming languages such as Spring and Angular.

He currently studies a Master's Degree at Loyola University, specializing in Artificial Intelligence.

LUCÍA GARCÍA LADO was born in Burgos, Spain in 2003. She received the degree in Computer Engineering from the University of Burgos, Spain, in 2025, with a specialization in Computing and Information Systems. She currently studies a Master's Degree at Loyola University, specializing in Artificial Intelligence.

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).