Metaheuristic Optimization for Efficient Food Production Scheduling

Aseel Abdelkareem[1], Rawan Hegazy[2], Ganna Moahmed[1], Jessica Magdy[2,3], and Omar M. Shehata[2,3]
[1]Computer Science Department, Faculty of Media Engineering and Technology, German University in Cairo, Egypt
[2]Mechatronics Engineering Department, Faculty of Engineering and Materials Science, German University in Cairo, Egypt
[3]Multi-Robot Systems (MRS) Research Group, Cairo, Egypt
{aseelabdelkareem, rowanhegazy2002, Gannamohamedsa}@gmail.com,
{jessica.gergis, omar.mohamad}@guc.edu.eg

***Abstract*— In this paper, we explore multiple metaheuristic optimization techniques for multi-cooperative production scheduling in the food industry. Our approach aims to minimize production costs by optimizing energy consumption, labor deployment, and raw material utilization while adhering to practical constraints such as demand satisfaction, production line exclusivity, where each line can produce only one product at a time, and line changeover time. Simulated Annealing (SA), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Social Spider Optimization (SSO) algorithms are implemented and evaluated on this problem. We validate the performance of each algorithm using two case studies, each representing contrasting scenarios in terms of demand and power consumption. Finally, all the results were analyzed and compared, highlighting the PSO algorithm's efficiency in generating cost-effective schedules compared to other algorithms. However, it came to be very computationally expensive. This study also demonstrates the potential of SSO in addressing complex production scheduling problems in manufacturing, as the results are similar to PSO with a slight difference.**

I. INTRODUCTION

Efficient production scheduling is a critical challenge in the manufacturing sector, especially in energy-intensive industries like food production. The increasing cost of energy and the growing environmental challenges require innovative solutions to optimize resource utilization and reduce operational costs. Traditional scheduling methods often struggle to balance competing objectives such as minimizing energy usage, meeting production demand, and managing labor efficiently.

Metaheuristic algorithms have emerged as powerful tools to solve such NP-hard scheduling problems due to their flexibility and ability to navigate large, complex search spaces. Techniques such as Simulated Annealing (SA) [7], Genetic Algorithms (GA)[5], Particle Swarm Optimization (PSO) [8], and Social Spider Optimization (SSO)[6] have been widely applied to a food production environment, where multiple production lines operate under constraints such as demand fulfillment, changeover time, and exclusivity.

This paper presents a comparative analysis of the four metaheuristic optimization algorithms, using two case studies: one representing a high-demand, high-power consumption scenario and another with low demand and power requirements. These scenarios demonstrate the versatility and with cost minimization and computational efficiency.

The remainder of this paper is organized as follows. Section II reviews the literature on production scheduling and optimization techniques. Section III describes the methodology and outlines the problem formulation in detail. Section IV presents the implementation of the optimization algorithms, including parameter settings and solution procedures. In Section V, we discuss the results obtained from algorithm validation and case studies. Finally, Section VI outlines the conclusion and future research directions.

II. LITERATURE REVIEW

Production scheduling problems are often addressed using Mixed-Integer Linear Programming (MILP), which effectively minimizes energy, labor, and production costs while considering constraints such as changeover times and exclusive line assignments [1]. However, MILP solutions may lack scalability and adaptability in dynamic environments.

In supply chain scheduling, both MILP and Mixed Integer Non-Linear Programming (MINLP) help meet customer demands with varying delay tolerances [2]. Metaheuristic algorithms such as Adaptive Genetic Algorithm (AGA) and Ant Lion Optimization (ALO) offer improved flexibility in resource allocation and better handling of tardiness penalties and lost sales.

For energy-efficient scheduling, swarm intelligence methods like Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) are widely adopted to reduce energy costs under constraints like peak usage [3]–[5]. These approaches outperform traditional methods in scalability and real-time adaptability, particularly with renewable energy integration.

In the context of the Food-Energy-Water (FEW) nexus, optimization models have been applied to urban systems to enhance resource efficiency and reduce environmental impacts [4]. Studies report up to a 21% reduction in resource use and a 29.1% decrease in GHG emissions, though implementation costs remain high.

Finally, multi-objective optimization techniques address several goals simultaneously—such as energy consumption, peak shaving, and makespan—by balancing trade-offs between objectives [5], [6]. These methods have shown success in complex industrial applications like automotive manufacturing and automated vehicle routing.

III. METHODOLOGY

A. Objective function

The presented objective function is the minimization of the energy, labor, and raw materials consumed to produce the

least cost over the scheduling horizon $\mathcal{T}$, which is formulated as:

$$\text{Total\_Cost} = \min \sum_{t=1}^{N_t} \sum_{l=1}^{N_l} (\lambda_t P_{lt} + \mu_t L_{lt} + \sum_{q=1}^{N_q} \beta_{qlt} Q_{ql} C_{ql}) \Delta t \tag{1}$$

Where the first and second terms represent the electricity consumption and labor costs, with P_{lt} and L_{lt} denoting the power consumption and labor requirements of the line l at time t; and the third term is the production cost of the product types in the production lines. λ_t and μ_t are, respectively, the electricity price and wage expense of the laborers at time t; C_{ql} is the unit production cost of the product type q manufactured on line l; and N_l and N_q designate the number of production lines and the number of product types, respectively. Assuming that the manufacturing line l consumes power at a constant rate P_l and requires a fixed number of laborers L_l during production, P_{lt} and L_{lt} are calculated respectively as:

$$P_{lt} = \alpha_{lt} P_l, \quad \forall l \in \mathcal{L}, \quad \forall t \in \mathcal{T}$$

$$L_{lt} = \alpha_{lt} L_l, \quad \forall l \in \mathcal{L}, \quad \forall t \in \mathcal{T}$$

B. Constraints

Demand Constraint: This constraint ensures that the total quantity of each product q produced across all lines and time intervals meets or exceeds the required demand D_q.

$$\sum_{l=1}^{N_l} \sum_{t=1}^{N_t} (\beta_{qlt} \cdot Q_{ql}) \geq D_q, \quad \forall q \in \{1, 2, \ldots, N_q\}$$

where Q_{ql} and D_q are given parameters, respectively specifying the unit quantity of the product q manufactured in line l and the total demand for the product q.

Exclusivity Constraint: This constraint ensures that each line l produces only one product at a time, or remains idle.

$$\sum_{q=1}^{N_q} \beta_{qlt} \leq \alpha_{lt}, \quad \forall l \in \{1, 2, \ldots, N_l\}, \forall t \in \{1, 2, \ldots, N_t\}$$

Changeover Condition: This constraint accounts for the off-time required T_{off} when switching production from one product to another on the same line.

$$\sum_{t'=t-T_{\text{off}}}^{t-1} (1 - \alpha_{lt'}) \geq T_{\text{off}} \left(\beta_{ql(t-1)} - \beta_{qlt}\right), \forall l \in \{1, 2, \ldots, N_l\}$$

C. Decision Variables

$$\alpha[l][t] = \begin{cases} 1 & \text{if line } l \text{ is active at time } t \\ 0 & \text{otherwise} \end{cases}$$
$$\forall l \in \{1, 2, \ldots, N_l\}, \forall t \in \{1, 2, \ldots, N_t\}$$

$$\beta[q][l][t] = \begin{cases} 1 & \text{if product } q \text{ is produced on line } l \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$
$$\forall q \in \{1, 2, \ldots, N_q\}, \forall l \in \{1, 2, \ldots, N_l\},$$
$$\forall t \in \{1, 2, \ldots, N_t\}$$

External Data:

$N_l = 3$ (Number of production lines)
$N_t = 5$ (Number of time periods)
$N_q = 2$ (Number of product types)
$\mu_t = [20, 22, 21, 23, 18]$ (Labor wages)
$P_l = [100, 120, 110]$ (Power consumption)
$L_l = [10, 12, 11]$ (Labor requirements)
$Q_{ql} = [[50, 60, 70], [30, 40, 50]]$ (Quantity of product)
$C_{ql} = [[1.5, 1, 2], [2.5, 4, 3]]$ (Cost of raw material)
$D_q = [200, 150]$ (Demand for each product type)
$T_{\text{off}} = 1$ (Changeover off-time)

D. Solution representation

Alpha Matrix: This matrix represents one possible configuration or solution, with 1s marking active elements or choices in the solution, and 0s marking inactive ones.
Beta Matrix:This matrix is a 3D array (suggested by the triple brackets) representing possibly multiple configurations or states related to the optimization.
Example for a solution:

$$\text{Alpha} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Beta} = \left[\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \right]$$

IV. Implementation

A. Simulated Annealing (SA) for Production Line Scheduling

The Simulated Annealing algorithm starts by initializing scheduling variables, cost parameters, and constraints. A random initial solution is evaluated, ensuring each production line is active for one or two periods and products are assigned only during active times. The algorithm then iteratively explores neighboring solutions that meet feasibility requirements such as demand satisfaction, exclusivity, and changeover time. New solutions are accepted if they yield a lower cost. If not, an acceptance probability is computed, and the solution is accepted if a random number falls below this probability; otherwise, it is rejected. The temperature is gradually reduced, and the solution is updated until a final temperature or maximum number of iterations is reached. This method helps escape local optima and aims to reach a cost-efficient schedule. Algorithm 1 outlines the main steps of this process.

Algorithm 1 Simulated Annealing for Scheduling

Initialize: decision variables α, β, cost parameters, constraints, T_{init}, cooling rate, max_iter
Compute cost of initial solution
while temperature ¿ threshold and iterations $< max_iter$ **do**
    Generate a valid random valid neighbor solution (α', β') satisfying all the constraints
    Compute cost of neighbor
    if cost is lower **then**
        Accept neighbor as current
    else
        Compute acceptance probability
        if random number ¡ probability **then**
            Accept neighbor
        else
            Reject neighbor
        end if
    end if
    Reduce temperature
end while
return best solution found

B. Genetic Algorithm (GA)

The Genetic Algorithm initializes a population of valid solutions, starting with 10 and increasing to 50, including scheduling variables α (for line activation) and β (for product allocation on lines). Fixed parameters include cost values (powerCost, μ_t, P_l, etc.) and constraints such as demand D_q and changeover time T_{off}. GA-specific parameters include a crossover rate of 0.7, an elite rate of 0.1, and a mutation rate of 0.2.

AEach generation, fitness is evaluated, elites are selected, and parents with the two lowest fitness values are chosen for single-point crossover on α and β matrices. Mutation flips a random bit in the remaining individuals. All offspring are checked for validity.

The process repeats until a termination condition (max generations or target fitness) is met. The algorithm returns the lowest-cost solution found.

Algorithm 2 Genetic Algorithm for Scheduling

Initialize: population of valid (α, β) schedules, cost parameters, constraints
Set GA parameters: crossover = 0.7, elite = 0.1, mutation = 0.2
while termination condition not met **do**
    Evaluate fitness of individuals
    Select elite and parents with lowest fitness
    Apply single-point crossover on (α, β)
    Mutate remaining individuals by flipping random bits
    Validate all new solutions
    Form next generation
end while
return individual with lowest fitness

C. Particle Swarm Optimization

Since the PSO algorithm is designed for continuous spaces, a continuous PSO version was used in [8]. To evaluate particle fitness, they performed a space transformation from a particle in the continuous space to a permutation in the solution space. We modified this approach to better suit binary problems by flattening α and β into 1D arrays. The swarm initializes with valid solutions, each particle having a position and velocity. Particles update their velocity based on inertia, personal best ($pBest$), and global best ($gBest$) positions, using random factors to encourage exploration. We used standard values: $w = 0.5$(inertia), $c_1 = 1.5$(cognitive), and $c_2 = 2$(social). Positions are updated and binarized via a sigmoid function to ensure feasibility. The process repeats until convergence, with $pbest$ and $gbest$ updated when better solutions are found.

Algorithm 3 Particle Swarm Optimization for Scheduling

Initialize: swarm of valid (α, β) solutions, flatten to 1D array
Set parameters: swarm size, max iterations, $w = 0.5$, $c_1 = 1.5$, $c_2 = 2$
for each particle **do**
    Initialize position and velocity randomly
    Evaluate fitness, set personal best $pBest$
end for
Set global best $gBest$ from best $pBest$
while termination condition not met **do**
    for each particle **do**
        Repair solution if constraints are violated
        Update velocity:

$$v \leftarrow wv + c_1 r_1 (pBest - position) + c_2 r_2 (gBest - position)$$

        Update position:

$$position \leftarrow position + v$$

        Apply sigmoid and binarize:

$$position = \mathbb{1}(\text{rand} < \sigma(v)), \quad \sigma(v) = \frac{1}{1 + e^{-v}}$$

        Update $pBest$ and $gBest$ if improved
    end for
end while
return global best position $gBest$

D. Social Spider Optimization

The Social Spider Optimization algorithm mimics the behavior of social spiders exploring and exploiting their environment. Each solution is seen as a spider that either explores randomly by flipping bits in the schedule preventing the algorithm from getting stuck in local optima and allows for a broader search, or exploits by moving towards better solutions by leveraging the best positions found so far. This step focuses on refining the solution ensuring the algorithm converges toward optimal or near-optimal solutions. We implemented this by considering the vibration intensity

produced from the best solution and comparing it to the current solution if the probability of the best solution is near to 1 we change the current solution's bit to match that of the best solution. The balance between exploration and exploitation is controlled by a predefined factor.

Algorithm 4 Social Spider Optimization for Scheduling

Initialize: population of valid (α, β) schedules
Set parameters: cost values, constraints, exploration factor
while termination condition not met **do**
  for each spider (solution) **do**
    Evaluate fitness
    if random ¡ exploration factor **then**
      Explore: randomly flip bits in (α, β)
    else
      Exploit: compare to best solution
      With probability $\sim$ 1, match bits with best solution
    end if
  end for
end while
return best-found solution

V. Results and Discussion

A. *SA Results*

Input Parameters:
• Initial temperature:1000, Final temperature:50
• D_q = [300, 100] (Lower demand for both product types).
• P_l = [100, 120, 110, 115, 130, 125, 105, 110, 140, 120] (Lower power consumption for production lines).
•Initial alpha and initial beta :

$$\alpha_{\text{init}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\beta_{\text{init}} = \begin{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{bmatrix}$$

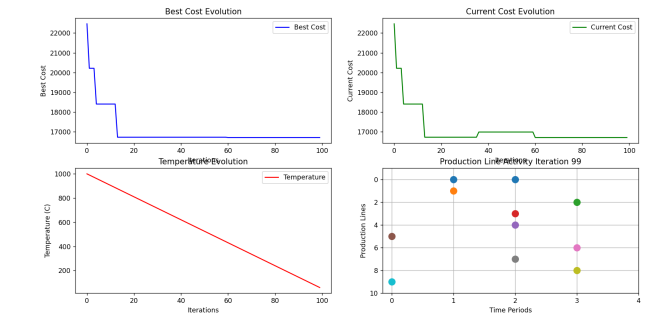

Fig. 1: SA Results

B. *GA Results and case studies*

Input Parameters:
• crossover rate = 0.7, mutation rate = 0.2, elite rate=0.1
• max iterations = 50 ,population size = 10

*1) **Case Study 1: High Demand and High-Power Consumption**:* Input Parameters:
• D_q = [500, 200] (Higher demand for both product types).
• P_l = [200, 240, 220, 230, 250, 240, 210, 220, 260, 240] (Higher power consumption for production lines).

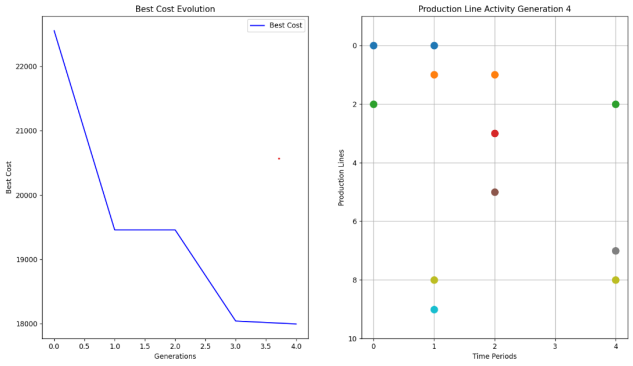

Fig. 2: GA case study 1 Results

*2) **Case Study 2: Low Demand and Low Power Consumption**:* Input Parameters:
• D_q = [100, 50] (Lower demand for both product types). • P_l = [50, 60, 55, 58, 65, 62, 52, 55, 70, 60] (Lower power consumption for production lines).

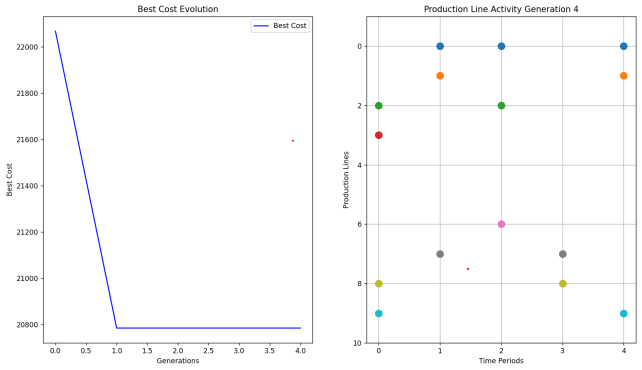

Fig. 3: GA case study 2 Results

Case Studies Analysis and Effect of the different algorithm parameters on its performance.

In Case Study 1, the GA showed significant cost improvement across generations, decreasing from 22,552.5 (Generation 1) to 17,995.5 (Generation 5) before stabilizing, while in Case Study 2, it converged quickly with minimal change. This indicates that the GA's performance is influenced by problem complexity higher demand and power require more exploration, whereas simpler scenarios stabilize faster with moderate parameters.

C. PSO Results

Input Parameters:
n-variables = 20, n-particles = 30, n-iterations= 100
w = 0.5 Inertia weight, c1 = 1.5 Cognitive coefficient, c2 = 2.0 Social coefficient

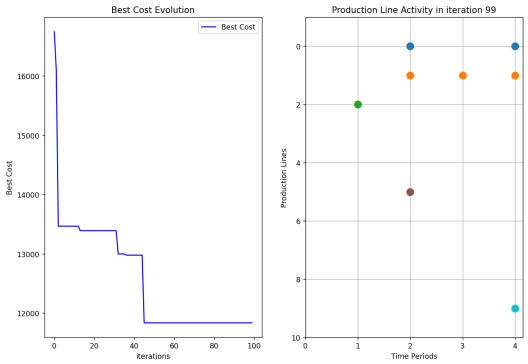

Fig. 4: PSO Results

The best cost was found to be 11832.

D. SSO Results and Case Studies

1) ***Case Study 1: Small Scale Configuration:*** *:* 10 production lines, 5 time periods, 2 product types, fewer variations of power consumption and labor requirements, decreased production capacities, and low raw material costs.[**?**]

2) ***High Scale Configuration:*** *:* 20 production lines, 5 time periods, 2 product types, higher variations of power consumption and labor requirements, increased production capacities, and high raw material costs.

The code for both configurations was run 5 times to analyze the average best costs. The analysis reveals that

Configuration	Run a	Run b	Run c	Run d	Run e
Small Scale	11,773.0	12,195.5	10,695.5	12,062.5	11,940.0
High Scale	16,986.0	17,701.0	18,221.0	14,316.0	14,445.0

TABLE I: Best Costs Across Runs for Small Scale and High Scale Configurations

Configuration	Average Best Cost
Small Scale Configuration	11,733.3
High Scale Configuration	16,333.8

TABLE II: Average Best Costs for Small Scale and High Scale Configurations

the input variables especially the number of production lines and associated resource requirements play a critical role in determining the best costs. Larger and more complex configurations, such as the first, inherently lead to higher costs due to increased resource consumption and operational complexity. Conversely, smaller-scale configurations, as in the second, achieve significantly lower costs. The results demonstrate the scalability and adaptability of the SSO algorithm across different problem sizes.

E. Algorithms Comparisons

Average Best Cost Comparison With 10 production lines, 5 time periods, and 2 products, we compared the average best cost of the four algorithms (SA, GA, PSO, SSO) running 5 times.

Algorithm	Average Best Cost
Simulated Annealing (SA)	20,468.5
Genetic Algorithm (GA)	13,627.8
Particle Swarm Optimization (PSO)	11,653.5
Social Spider Optimization (SSO)	12,381.9

TABLE III: Average Best Costs for Different Algorithms

Based on the average best costs, PSO emerged as the most effective algorithm in terms of cost minimization, followed by SSO, GA, and SA. This analysis suggests that PSO is better suited for this particular optimization problem due to its ability to explore the solution space effectively while converging to a lower cost. However, factors such as runtime and convergence behavior must be studied since they must be taken into consideration when selecting an optimization algorithm for a given problem. Simulated Annealing (SA), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Social Spider Optimization (SSO) are popular metaheuristic algorithms, each with distinct strengths and weaknesses. SA is a trajectory-based algorithm inspired by the annealing process in metallurgy, effective for escaping local optima but often limited by its slow

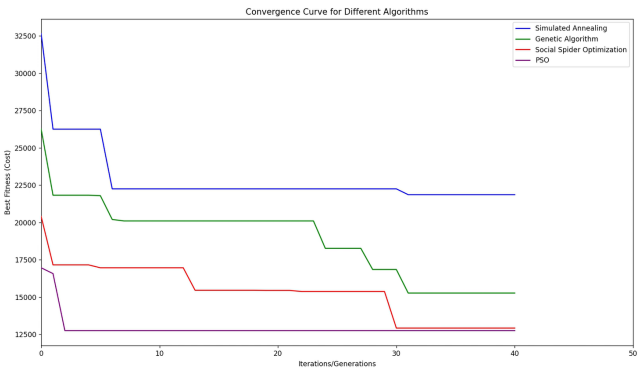

Fig. 5: Convergence curve

convergence in large solution spaces. GA, inspired by the principles of natural selection, utilizes crossover and mutation to explore a wide solution space and is well-suited for diverse optimization problems, but it can suffer from premature convergence without proper parameter tuning. PSO, based on the social behavior of bird flocks or fish schools, efficiently balances exploration and exploitation by updating solutions through velocity adjustments, yet it may converge prematurely in complex landscapes. SSO, derived from spider social behavior, excels in leveraging cooperative interactions to search the solution space comprehensively and adapt to constraints, making it particularly effective in complex, multi-objective problems. Among these, SSO often demonstrates superior performance in terms of cost minimization and constraint handling in scheduling tasks, as it effectively balances exploration and exploitation, whereas SA, GA, and PSO may require more extensive parameter fine-tuning to achieve comparable results.

Algorithm	Avg Best Cost	Optimal Cost	Std Dev	Avg time(s)
SA	20468.50	17778.50	2426.15	**2.63**
GA	13627.80	12683.50	2181.53	13.08
PSO	**11653.50**	**11447.50**	**187.75**	78.63
SSO	12381.90	11702.00	569.04	31.11

TABLE IV: Performance comparison of optimization algorithms over 5 runs based on average best cost, optimal cost, standard deviation, and average runtime.

VI. Conclusion and Future Recommendations

In conclusion, efficient production scheduling remains a crucial factor in achieving sustainability and cost-effectiveness in energy-intensive industries such as food production. This study proposed a novel optimization framework based on the Social Spider Optimization (SSO) algorithm to address the challenges of balancing energy consumption, labor utilization, and raw material usage while meeting practical constraints. By testing the framework in two distinct case studies, we demonstrated the effectiveness of SSO in generating superior schedules compared to traditional metaheuristic algorithms like Genetic Algorithm (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). The results underline SSO's potential as a robust solution for complex production scheduling problems, paving the way for its broader adoption in manufacturing environments.

Future research could explore several directions and algorithms to enhance and extend the proposed framework. First, incorporating real-time scheduling capabilities would enable dynamic adjustments to unexpected disruptions, such as machine breakdowns or fluctuating demand. Second, integrating renewable energy sources into the scheduling model could further optimize energy costs while promoting sustainability. Additionally, combining SSO with hybrid approaches or other advanced metaheuristics may further improve solution quality and computational efficiency. Finally, applying the framework to other manufacturing sectors with unique scheduling challenges could validate its versatility and expand its applicability across diverse industries.

References

[1] F. Angizeh, H. Montero, A. Vedpathak, and M. Parvania, "Optimal production scheduling for smart manufacturers with application to food production planning," Computers & Electrical Engineering, 2020.

[2] S. Aminzadegan, M. Tamannaei, and M. Rasti-Barzoki, "Multi-agent supply chain scheduling problem by considering resource allocation and transportation," Computers & Industrial Engineering, vol. 137, 2019.

[3] K. Gao, Y. Huang, A. Sadollah, et al., "A review of energy-efficient scheduling in intelligent production systems," Complex Intell. Syst., vol. 3, 2019.

[4] P. Zhang, L. Zhang, Y. Hao, M. Xu, M. Pang, C. Wang, A. Yang, and A. Voinov, "Food–energy–water nexus optimization brings substantial reduction of urban resource consumption and greenhouse gas emissions," PNAS Nexus, vol. 3, 2024.

[5] T. Küster, P. Rayling, R. Wiersig, et al., "Multi-objective optimization of energy-efficient production schedules using genetic algorithms," Springer Link, vol. 3, 2021.

[6] E. Cuevas, M. Cienfuegos, D. Zaldívar, and M. Pérez-Cisneros, "A swarm optimization algorithm inspired in the behavior of the social-spider," Expert Systems with Applications, vol. 40, no. 16, pp. 6374–6384, 2013.

[7] K. Wang, X. Li, L. Gao, P. Li, and S. M. Gupta, "A genetic simulated annealing algorithm for parallel partial disassembly line balancing problem," *Computers & Industrial Engineering*, 2020.

[8] W. Pang, K. P. Wang, C. G. Zhou, and L. J. Dong, "Fuzzy discrete particle swarm optimization for solving traveling salesman problem," *Proceedings of the 4th International Conference on Computer and Information Technology (CIT'04)*, 2004.