

A Metaheuristic Approach to Job Shop Scheduling Using Cat Swarm Optimization Algorithm

Mahesh S. Shinge^{1*} and Sagar U. Sapkal¹

¹Walchand College of Engineering, Sangli, Shivaji University, Kolhapur, Maharashtra, India

Abstract. Cat Swarm Optimization (CSO) is a metaheuristic algorithm that is inspired by cat behavior. While solving the optimization problem, the seeking and tracing modes of CSO balance the exploration and exploitation. This study developed the CSO algorithm to address the Job Shop Scheduling Problem (JSSP). JSSP is the assignment of tasks to available machines. The objective of scheduling is to reduce the overall completion duration, considering the precedence constraint. Efficient scheduling is crucial in manufacturing and production planning. In this research paper, the CSO algorithm framework is integrated with an active schedule strategy. By transforming candidate solutions into active schedules in each iteration, the algorithm investigates only significant areas of the search space and avoids infeasible or poor solutions. The proposed metaheuristic is evaluated on Lawrence instances. Results demonstrate that the CSO with an active schedule strategy improves solution quality and convergence rate relative to the basic PSO methods. This approach gives an effective way to address JSSP in the manufacturing sector.

1 Introduction

Job shop scheduling represents one of the most challenging issues in the domain of industrial optimization. It has been studied in operations research since the field began. The Job Shop Scheduling Problem (JSSP) seeks to determine the optimal sequence of tasks for various jobs across multiple machines. Each job has a fixed sequence and specific processing needs. JSSP is classified as NP-hard [1]. This means that solving large-scale problems exactly is not practical within a reasonable time. The primary objective in JSSP is to minimize the makespan, defined as the overall time required to finish all tasks. This goal reflects the growing complexity of modern production systems. Traditional methods use mathematical programming and exact algorithms [2]. However, these methods face limits because their computation grows very fast with problem size. Metaheuristic approaches provide an alternative. They give near-optimal solutions in less time. Swarm intelligence methods are part of these approaches and have been studied widely for JSSP. They copy the behavior of natural systems to explore solution spaces effectively. Nature-inspired computational methodologies such as Genetic Algorithms [3] and Particle Swarm Optimization [4] have

* Corresponding author: maheshs.shinge@gmail.com

been applied successfully. In many domains, Cat Swarm Optimization (CSO) has found applications, including engineering optimization and scheduling. Studies show that CSO can find better solutions than some traditional techniques and also converge faster [5]. CSO is flexible, and its operators can be modified or combined for better performance in different domains. Feasible schedule generation is an important part of JSSP. Among viable solutions, active schedules are the most frequently utilized. In an active schedule, starting one operation is impossible without causing a delay to another. Active schedules guarantee that an optimal solution exists within them [6]. For this reason, many optimization algorithms focus on active solutions. Active decoding is often used to transform algorithm outputs into implementable active schedules. This ensures that generated schedules follow problem constraints and make efficient use of resources. Modern manufacturing requires adaptive scheduling methods. This study demonstrates how CSO can be combined with active schedule construction to obtain better solutions. The objective is to develop a hybrid approach that uses CSO's exploration ability with the structure of active solutions. The remaining sections of this paper present the mathematical model, algorithm design, experimental results, and comparison with other methods. The study emphasizes how nature-inspired optimization methods can support difficult scheduling problems and guide future research in intelligent production systems.

2 Related work

The fundamental Cat Swarm Optimization (CSO) algorithm, first presented by Chu et al. [7]. The approach has two modes to replicate the inherent behavior of cats, namely seeking and tracing. They demonstrated that the CSO algorithm outperformed PSO across several benchmark instances [7]. Orouskhani et al. solved an unconstrained optimization problem by a modified CSO algorithm with adaptive parameters. The algorithm is modified by using adaptive coefficients for inertial weight and acceleration. The unique update rule for the current position was developed. The authors found that these adaptations improved the speed of convergence and solution quality [8]. Gabi et al. integrated a Simulated Annealing algorithm with a CSO to schedule tasks in a cloud environment. An orthogonal Taguchi design was employed to facilitate scalability. Their results provided higher service quality in comparison to other scheduling algorithms [9]. Khedim et al. developed an Artificial Bee Colony algorithm for JSSP. In this study, a rank-based selection strategy with position-based crossover is used for updating solutions during the onlooker bee phase. [10]. An upgraded version of the CSO was developed by Songyang et al. They incorporated an information top-N learning strategy and interaction strategy into the original CSO algorithm [11]. Bouzidi and Riffi introduced a discrete version of the CSO for the JSSP [12]. The same authors addressed the open shop scheduling problem by refining the representation method for the position and velocity functionality of the CSO [13]. Ahmed et al. carried out a detailed survey of the CSO. This study focused on the variations and applications of CSO. The authors discussed recent developments in hybrid approaches [5]. Mahmud et al. developed a hybrid evolutionary algorithm by integrating Differential Evolution and PSO for JSSP [14]. The fuzzy-weighted NSGA-II was developed by Delgoshai et al. to handle multi-objective JSSP. The proposed approach considered uncertain conditions in decision-making, making a better compromise between objectives [15]. Khurshid and Maqsood proposed a hybridization of the Evolution algorithm and Simulated Annealing algorithm to solve the JSSP [16].

3 Job Shop Scheduling Problem

3.1 Mathematical formulation

The principal aim of the JSSP is to reduce the overall duration of the schedule, referred to as makespan. A job shop is characterized by a set of n jobs and m machines. Each job has a predefined operation sequence that is required to be executed on a specific machine for a defined duration in time. The makespan is reduced by determining the feasible schedule of operations over the machines, satisfying the constraints. A feasible schedule will ensure that whenever two operations on a single job have a precedence relationship, they are scheduled in the correct order. No two or more operations should be processed by a machine simultaneously; therefore, all operations must be executed in the predefined order [17]. There are a total of $m \times n$ operations. To model JSSP, dummy operations 0 and $n \times m + 1$ with zero processing time are considered. An operation set is defined as, $O = \{0, 1, 2, \dots, n \times m + 1\}$. Any operation can be started after the completion of its predecessor operations P_i .

$$\text{Minimize } C_{\max} \quad (1)$$

$$C_q \leq C_i - t_i, \quad i = 0, 1, 2, \dots, n \times m + 1; \quad q \in P_i \quad (2)$$

$$\sum_{i \in A(t)} \omega_{im} \leq 1, \quad m \in M, \quad t \leq 0 \quad (3)$$

$$C_i \geq 0, \quad i = 0, 1, 2, \dots, n \times m + 1 \quad (4)$$

Equation (1) specifies the objective function of JSSP, which is the minimization of the makespan. Equation (2) specifies the precedence relations constraints. Equation (3) specifies the constraint that at any machine, only one operation can be executed. Equation (4) implies the positive completion time.

3.2 Active Schedule

There are an infinite number of feasible schedules for the JSSP. Figure 1 shows the types of schedules. A semi-active schedule is a subset of all feasible schedules where the earlier starting of any operation without changing the order of operations on the same machine is impossible. An active schedule is a subset of semi-active schedules. An active schedule is a schedule where any operation cannot commence earlier without causing delay to another operation or breaking feasibility. A machine does not remain idle while an operation is waiting [6].

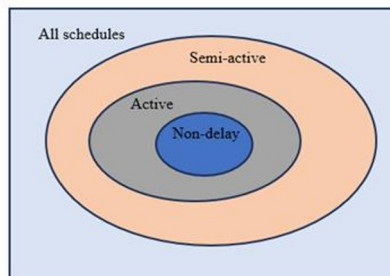


Fig. 1. Types of schedules.

An active schedule ensures that all jobs are arranged in a way that respects machine availability and job order while avoiding unnecessary idle times. Active schedules are important because the set of all active schedules contains at least one optimal solution. This means we do not need to consider every possible schedule, which reduces the search space. In practice, generating an active schedule helps in balancing efficiency and feasibility. It avoids wasted machine time while keeping the job sequence consistent with constraints. For this reason, many algorithms for JSSP use active schedule generation as part of their decoding or solution-building.

4 Cat swarm optimization algorithm with active schedule for JSSP

4.1 CSO Algorithm

Chu et al. (2006) developed the swarm-based Cat Swarm Optimization (CSO) algorithm [7]. CSO simulates certain behaviors of cats. In CSO, each cat is associated with a candidate solution in an M-dimensional space. CSO has two modes of behavior: the first is the seeking mode (resting/observing), and the second is the tracing (chasing a target) mode. In seeking mode, a cat is resting in alert mode by sampling the positions nearby, while in tracing mode, it will run toward what it sees as the best position. These modes are mixed with a user-defined mixture ratio (MR) that determines the proportion of how many cats go into tracing mode. The authors tested the results of the benchmark tests on six standard functions, showing that Cat Swarm Optimization always found better results than both basic PSO and a PSO with an inertia-weight (weighted PSO). The flowchart for CSO is illustrated in Figure 2. Sections 4.1.1 and 4.1.2 describe the procedure of CSO. The results suggest that the new cat swarm optimization can converge to the optima better than the PSO versions tested [7].

4.1.1 Seeking Mode

Most of the time, cats are at rest with eyes open, tracking their surroundings. In this seeking mode, cats are predominantly inactive even when awake; they prefer to rest, but they remain alert to movement. The Seeking mode is intended as a local exploration, i.e., neighborhood search. It has the following parameters. SMP (Seeking Memory Pool) refers to the number of candidates that shall be produced for each cat. The maximum percentage change permitted when modifying a dimension is defined by SRD (Seeking Range of Dimension). The number of dimensions to perturb for each candidate is defined by CDC (Counts of Dimensions to Change), and whether the cat's present position is a candidate is indicated by Boolean SPC (Self-Position Consideration). Following is the stepwise procedure of seeking the mode [7].

Step 1: Based on the current position of the cat, SMP copies are created. For the true SPC, one of these copies is the cat's current position. Otherwise, all copies of current position are the altered version.

Step 2: The CDC dimensions for each copy are selected. Randomly increase or decrease the values of these dimensions by a percentage defined by SRD.

Step 3: For each copy, evaluate the fitness value.

Step 4: If the fitness values of all candidates are unequal, assign the probability for selection to each candidate using equation 1. If all are equal, assign probability 1 to each candidate.

Step 5: Randomly select one candidate using the probabilities. Probability is calculated by equation (5). The cat now has a new position for the next iteration. For the goal of minimizing the function use $\text{Fitness}_b = \text{Fitness}_{\max}$ otherwise $\text{Fitness}_b = \text{Fitness}_{\min}$

$$P_i = \frac{|\text{Fitness}_i - \text{Fitness}_b|}{\text{Fitness}_{\max} - \text{Fitness}_{\min}} \quad (5)$$

4.1.2 Tracing Mode

In CSO, cats in tracing mode become active hunters and move toward prey with quickness and agility. The tracing mode emulates a cat that chases a target. The following are the steps to update the cat [7].

Step 1: For each dimension of the cat's position, update the velocity using Equation (6) that moves the cat toward the best-known solution. Set the limit range for velocity. If the velocity is higher than the set limit, then define it as the maximum velocity.

Step 2: The next position of the cat is updated by applying Equation (7)

$$\text{Velocity}_{k,d+1} = \text{Velocity}_{k,d} + r_1 \times c_1 \times (x_{\text{best},d} - x_{k,d}), \text{ where } d = 1, 2, \dots, M \quad (6)$$

$$x_{k,d+1} = x_{k,d} + \text{Velocity}_{k,d+1} \quad (7)$$

In Equations (2) and (3), $x_{k,d}$ is the current position of the cat and $x_{\text{best},d}$ is the best cat position. $V_{k,d+1}$ is the updated velocity and $x_{k,d+1}$ is the updated position.

4.1.3 CSO Algorithm for JSSP.

During the optimization process, initially, parameters are set. A random initial population of cats is generated with velocities. In every iteration, the mode of each cat is decided. During the seeking phase, potential solutions are generated in proximity to the current position, and the optimal best is chosen. Position and velocity are updated to the global best when in tracing mode. All the solutions will be transformed into active schedules. If an improvement is made by one of the solutions, the global best is updated. This process is repeated for a predetermined maximum number of iterations is reached. The most efficient schedule is returned at the conclusion of the process. Pseudocode for CSO for the JSSP algorithm is represented as follows. The flowchart is shown in Figure 2.

Algorithm 1 CSO for the JSSP

Initialize parameters:

N(population size): Number of cats, M: Problem dimension, MR, SMP, SRD, Vmax: Maximum velocity allowed in tracing mode, IterMax: Maximum number of iterations

Generate random positions X_i , velocities V_i for all N cats within the search space.

for iter = 1 to termination criteria do (termination criteria is iteration < IterMax):

for each cat i in the population:

assign the mode of cat i:

- cat i enters tracing mode with probability MR,
- cat i enters seeking mode with probability (1 - MR),

if cat i is in seeking mode:

if SPC = true:

generate SMP candidate positions around X_i within the range SRD.

ensure candidates remain within [PositionMin, PositionMax]

transform candidates → active schedule

else:

generate (SMP - 1) candidate positions around X_i within the range SRD.

ensure candidates remain within [PositionMin, PositionMax].

include X_i itself as one candidate.

transform candidates \rightarrow active schedule
calculate the fitness of all solutions.
define the best solution as the new X_i .
else if cat i is in tracing mode:
update velocity V_i using the global best position found so far.
clamp velocity components to $[-V_{max}, +V_{max}]$.
update position $X_i \leftarrow X_i + V_i$.
transform candidates \rightarrow active schedule
evaluate all the candidates' fitness.
if a better solution is found, **update** the global best position.
output the most efficient solution.

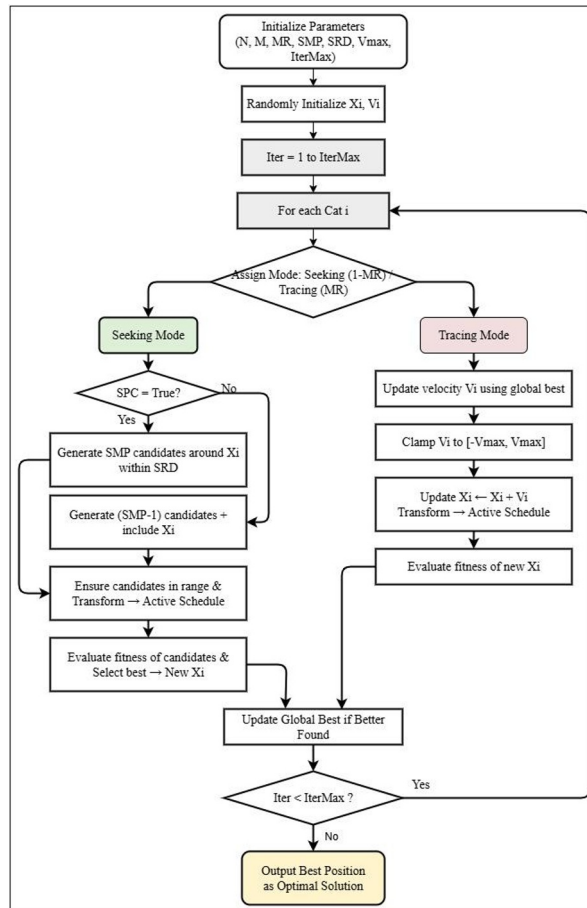


Fig. 2. Flowchart for CSO

4.2 JSSP Solution representation

The JSSP consists of n jobs and m machines. The $n \times m$ vector represents the solution. Table 1 shows a three-job and three-machine JSSP. This problem is specified as a 3×3 JSSP. Every job involves three operations, all of which are performed on designated machines. There are three jobs: J1, J2, and J3. The machines are designated M1, M2, and M3. T represents the time required to execute the task. The third operation of the second job is denoted as O23.

Where O is the operation, 2 represents the job number, whereas 3 denotes the third operation. Similarly, O12 denotes the second operation of the first job.

Table 1. 3 jobs and 3 machines (3×3) JSSP

| Job | Operation | Machine | T |
|-----|-----------|---------|----|
| J1 | O11 | M1 | 07 |
| | O12 | M3 | 04 |
| | O13 | M2 | 02 |
| J2 | O21 | M2 | 10 |
| | O22 | M1 | 01 |
| | O23 | M3 | 01 |
| J3 | O31 | M3 | 07 |
| | O32 | M1 | 05 |
| | O33 | M2 | 02 |

4.2.1 Encoding and decoding of a solution

A solution is usually represented as a sequence of operations. Each job is broken into its operations and presented in a list. The order of occurrence of each job's number shows the operation number by sequence. For example, job will appear for two times in the list if it has two operations. Appearance for each time indicating the next operation of that job. This representation is termed the operation-based representation. The solution generated by this method is compact and easy for algorithms to handle. Decoding means converting the operation sequence into a real schedule on machines. During decoding, each operation is placed on its assigned machine. The operation start time is decided by considering two constraints. First, it cannot start before its job's previous operation finishes, and second, it cannot start before the machine becomes free. A complete and feasible schedule is generated by considering these constraints. This coding and decoding process helps algorithms explore solutions effectively. This coding and decoding process helps algorithms explore solutions effectively. Coding gives a simple representation to work with, while decoding ensures that the solution is valid in the real scheduling environment.

4.2.2 Initial solution

Consider 3×3 JSSP from Table 1. For initial solution generation, first, random numbers are assigned to each operation as shown in Table 2. Then operations are arranged by sorting random numbers from smallest to largest. The vector (2, 3, 1, 1, 3, 2, 3, 2, 1) indicates the job operations sequence. In this representation, number 2 represents the operation number sequentially of the second job. Thus, the generated solution is O21-O31-O11-O12-O32-O22-O33-O23-O13. Figure 3 shows the Gantt chart for the given problem. Consider the scheduling of the operations of job 3. The scheduling of job 3 follows the required operation sequence and machine availability conditions. The first operation of job 3 is assigned to machine 3. Since machine 3 is free at the selected time and job 3 has the highest priority in the initial solution vector, this operation is started immediately. Once the first operation finishes, the second operation must be processed on machine 1. Machine 1 completes the first operation of job 1 at time 7, making it available exactly when job 3 requires it. Therefore, the second operation of job 3 is scheduled on machine 1 at time 7. The third operation of job 3 requires machine 2. Machine 2 finishes the first operation of job 2 at time 10, so it becomes idle at that moment. However, the third operation of job 3 cannot begin until the second operation is completed. Since the second operation finishes at time 12, the earliest feasible start time on machine 2 is also 12. As a result, the third operation is scheduled at time 12,

ensuring that the precedence constraints within job 3 are fully satisfied. This procedure always generates feasible solutions only. Before the evaluation of fitness value, each generated solution is transformed into an active solution.

Table 2. Assignment of Random numbers to each operation

| Job | 1 | | | 2 | | | 3 | | |
|---------------|---|----|----|----|----|---|---|----|----|
| Operation | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
| Random number | 9 | 11 | 37 | 31 | 23 | 3 | 7 | 29 | 21 |

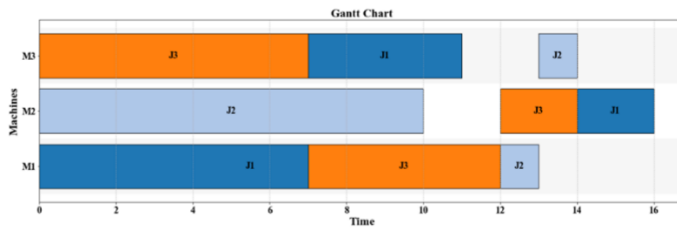


Fig. 3. Gantt chart for 3×3 JSSP

4.2.3 Active schedule generation

During the initial solution generation and in every iteration of the CSO process for JSSP, multiple solution candidates are created. All generated solutions are feasible and are then converted into active schedules using a simple procedure. A straightforward algorithm is developed to convert any schedule to an active schedule. Every operation is tested for a possible left shift without postponing another operation. If the left shift of any operation is possible by considering the precedence constraint, it is shifted, and the final solution is generated. From Figure 3, it is clearly observed that the third operation of the first job can be started early, as there is an ideal slot on machine 2. In this slot, this operation can be scheduled without violating the precedence constraint and without affecting any other operation. After rescheduling, the solution becomes an active schedule. Due to rescheduling, only the sequence of operations on machine M2 will be changed from O21-O33-O13 to O21-O13-O33. But it does not affect the completion time of any other operation. The remaining schedule remains unchanged. The solution vector is (2, 3, 1, 1, 3, 2, 1, 3, 2). Figure 4 demonstrates the Gantt chart. This procedure to generate an active schedule is explained in Algorithm 2. The flowchart for the active solution generation algorithm is represented in Figure 5. Only active schedules are considered and evaluated throughout the optimization process.

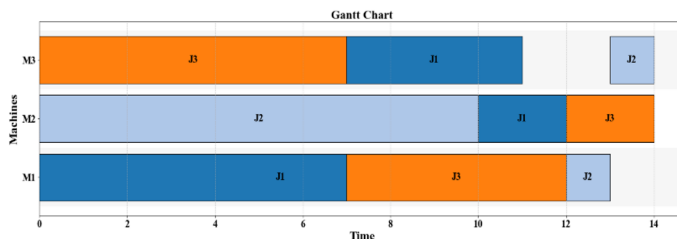


Fig. 4. Gantt chart for Active Schedule

Algorithm 2 Active Schedule Generation

Initialization:

```
for each machine m:  
    MachineTimeline[m] = empty  
for each job j:  
    CompletionTime[j] = 0  
Schedule = empty    m - machine assigned to operation o  
    p - processing time of operation o  
    // Determine precedence constraint  
    if o has no predecessor, then  
        PrecedenceTime = 0  
    else  
        PrecedenceTime = finish time of predecessor operation  
    // Find earliest feasible start time on machine m  
    StartTime = earliest gap on MachineTimeline[m]  
        that begins  $\geq$  PrecedenceTime  
        and is at least p long  
    if no gap exists then  
        StartTime = end of MachineTimeline[m]  
        FinishTime = StartTime + p  
    // Record operation  
    add (o, StartTime, FinishTime) to Schedule  
    insert (StartTime, FinishTime) into MachineTimeline[m] in sorted order  
    CompletionTime[job(o)] = FinishTime  
end for  
Output: Schedule
```

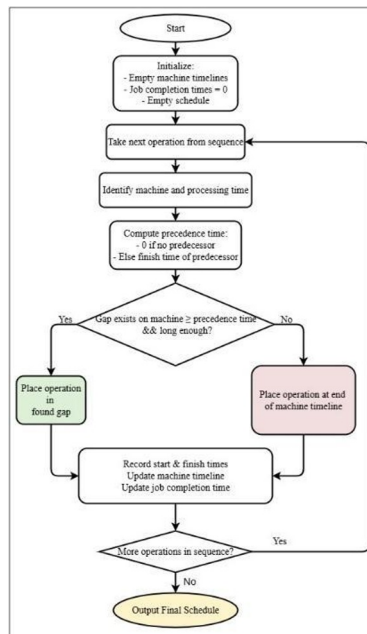


Fig. 5. Flowchart for the Algorithm for Active Schedule Generation

5 Results and Discussion

The proposed CSO algorithm was tested on twenty Lawrence benchmark problems from the OR-Library [18]. These problems included between 10 to 20 jobs and 5 to 10 machines. Each instance was run for 10,000 iterations, and the best result from fifty independent runs was recorded. The parameters of CSO are tuned as $N = 30$, $MR = 0.8$, $SMP = 5$, $CDC = 0.3$, $C1 = 2.0$. The performance was compared with two versions of particle swarm optimization: PSO1 (our implementation) and PSO2 [19]. The comparison was made using the Average Relative Percentage Error (ARPE) with respect to the best-known solution. ARPE provides a normalized comparison with best-known solutions. ARPE for 'I' number of instances is calculated by Equation (8). Where BKS_i and BS_i are the best-known solution and best solution obtained for the instance, respectively.

$$ARPE = \sum_{i=1}^I \left(\frac{BS_i - BKS_i}{BKS_i} \right) \times 100 / I \quad (8)$$

The results show that CSO performed very well across most of the test instances. For smaller problems such as La1 to La15, CSO consistently reached the BKS with zero relative error. PSO1 and PSO2 also obtained the BKS in many of these small problems, but PSO2 produced higher errors when compared with CSO. In larger problem instances (La16 through La20), the distinctions between the algorithms became much clearer. Utilizing CSO resulted in a very low ARPE value of 0.14. PSO1 and PSO2 yielded an ARPE value of 0.27 and 2.60, respectively.

Table 3. Computational Results

| La Instances | J × M | BKS | CSO | PSO1 | PSO2 | RPE CSO | RPE PSO1 | RPE PSO2 |
|--------------|---------|------|------|------|------|---------|----------|----------|
| 1 | 10 × 5 | 666 | 666 | 666 | 666 | 0.00 | 0.00 | 0.00 |
| 2 | | 655 | 655 | 655 | 655 | 0.00 | 0.00 | 0.00 |
| 3 | | 597 | 597 | 604 | 624 | 0.00 | 1.17 | 4.52 |
| 4 | | 590 | 590 | 590 | 627 | 0.00 | 0.00 | 6.27 |
| 5 | | 593 | 593 | 593 | 593 | 0.00 | 0.00 | 0.00 |
| 6 | 15 × 5 | 926 | 926 | 926 | 926 | 0.00 | 0.00 | 0.00 |
| 7 | | 890 | 890 | 890 | 890 | 0.00 | 0.00 | 0.00 |
| 8 | | 863 | 863 | 863 | 863 | 0.00 | 0.00 | 0.00 |
| 9 | | 951 | 951 | 951 | 951 | 0.00 | 0.00 | 0.00 |
| 10 | | 958 | 958 | 958 | 958 | 0.00 | 0.00 | 0.00 |
| 11 | 20 × 5 | 1222 | 1222 | 1222 | 1222 | 0.00 | 0.00 | 0.00 |
| 12 | | 1039 | 1039 | 1039 | 1039 | 0.00 | 0.00 | 0.00 |
| 13 | | 1150 | 1150 | 1150 | 1150 | 0.00 | 0.00 | 0.00 |
| 14 | | 1292 | 1292 | 1292 | 1292 | 0.00 | 0.00 | 0.00 |
| 15 | | 1207 | 1207 | 1207 | 1207 | 0.00 | 0.00 | 0.00 |
| 16 | 10 × 10 | 945 | 946 | 959 | 962 | 1.48 | 1.48 | 1.80 |
| 17 | | 784 | 784 | 787 | 822 | 0.00 | 0.38 | 4.85 |
| 18 | | 848 | 848 | 848 | 857 | 0.00 | 0.00 | 1.06 |
| 19 | | 842 | 842 | 852 | 891 | 0.00 | 1.19 | 5.82 |
| 20 | | 902 | 914 | 912 | 1152 | 1.33 | 1.11 | 27.72 |
| ARPE | | | | | | 0.14 | 0.27 | 2.60 |

Figure 6 plots the RPE for each instance and algorithm. It clearly illustrates that CSO's error stays near zero across almost all cases, whereas PSO1 and PSO2 have noticeably larger errors. This comparison confirms that CSO consistently yields the best solutions. Figure 7 shows representative convergence curves of the CSO algorithm for two sample instances (La2 and La9). These plots demonstrate that CSO quickly approaches the best-known makespan. The makespan value rapidly decreases in early iterations and then levels off as the algorithm converges. This indicates strong convergence speed and stability of CSO in practice. In general, the results indicate that CSO is stable and reliable. Figure 8 represents a Gantt chart for sample instance La15. Using the active schedule decoding strategy in the CSO methodology is a main contributor to the better-performing algorithm, but it was also particularly advantageous in the more complex job shop scheduling problem instances.

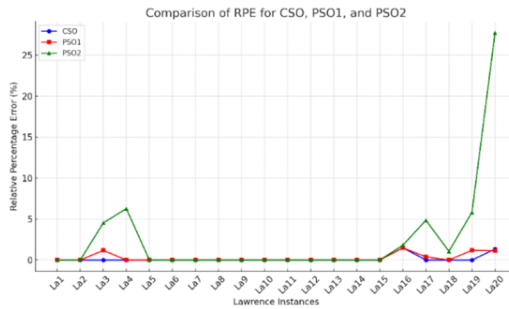


Fig. 6. Comparison of RPE for CSO, PSO1, and PSO2

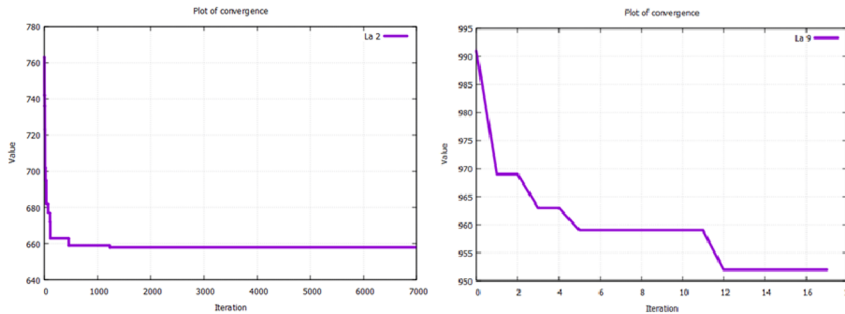


Fig. 7. Convergence Chart for La2 and La9 instances.

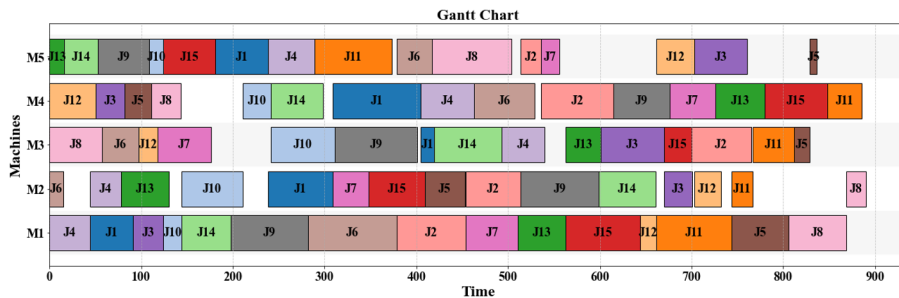


Fig. 8. Gantt Chart for La15 instance.

6 Conclusion

The suggested CSO algorithm was evaluated on the standard set of Lawrence benchmark problems. Its performance is compared with the two PSO algorithms. The results demonstrate that CSO represents a substantial enhancement compared to PSO approaches. CSO achieved near-optimal solutions with minimal error rates, but the PSO approaches exhibited significantly greater error rates. This suggests that the CSO is promising for job shop scheduling challenges. CSO generates reliable and high-quality solutions while maintaining an adequate level of robustness. This study has some limitations. The assessment was confined to a limited set of Lawrence benchmark problems. The evaluation focused solely on minimizing makespan. Tardiness, setup time, or machine failures are not included here. Future research may concentrate on analyzing CSO by considering more extensive datasets associated with real-world contexts. CSO can be expanded to analyze other objectives, including minimizing lateness or enhancing machine utilization. CSO can be hybridized with other metaheuristics to enhance the quality of the solution and convergence speed.

References

1. J. K. Lenstra, A. R. Kan, P. Brucker, Complexity of machine scheduling problems., *Annals of discrete mathematics*, 1, 343–362, (1977)
2. S. Ashour, S. R. Hiremath, A branch-and-bound approach to the job-shop scheduling problem, *International Journal of Production Research*, 11, 47–58, (1973), <https://doi.org/10.1080/00207547308929945>
3. S. Tian, C. Zhang, J. Fan, X. Li, L. Gao, A genetic algorithm with critical path-based variable neighborhood search for distributed assembly job shop scheduling problem., *Swarm and Evolutionary Computation*, 85, 101485, (2024), <https://doi.org/10.1016/j.swevo.2024.101485>
4. D. B. Fontes, M. M. Homayouni, S.M., J. F. Gonçalves, A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources., *European Journal of Operational Research*, 306, 1140–1157, (2023), <https://doi.org/10.1016/j.ejor.2022.09.006>
5. A. M. Ahmed, T. A. Rashid, S. Saeed, Cat Swarm Optimization Algorithm: A Survey and Performance Evaluation, *Computational Intelligence and Neuroscience*, **2020**, 1–20, (2020), <https://doi.org/10.1155/2020/4854895>
6. K. R. Baker, D. Trietsch, *Principles of sequencing and scheduling*, (John Wiley & Sons, 2018)
7. S. C. Chu, P. Tsai, J. S. Pan, Cat Swarm Optimization, *PRICAI 2006: Trends in Artificial Intelligence*, 4099, 854–858, (2006), https://doi.org/10.1007/978-3-540-36668-3_94
8. M. Orouskhani, Y. Orouskhani, M. Mansouri, M. Teshnehlab, A novel cat swarm optimization algorithm for unconstrained optimization problems, *International Journal of Information Technology and Computer Science*, 5, 32–41, (2013)
9. D. Gabi, A. S. Ismail, A. Zainal, Z. Zakaria, A. Khasawneh, Hybrid cat swarm optimization and simulated annealing for dynamic task scheduling on cloud computing environment, *Journal of Information and Communication Technology*, 17, 435–467, (2018)

10. A. O. Khedim, M. Souier, Z. Sari, Combinatorial artificial bee colony algorithm hybridised with a new release of iterated local search for job shop scheduling problem, *International Journal of Operational Research*, 44, 435, (2022)
11. L. Songyang, Y. Haipeng, W. Miao, Cat swarm optimization algorithm based on the information interaction of subgroup and the top-N learning strategy, *Journal of Intelligent Systems*, **31**, 489–500, (2022), <https://doi.org/10.1515/jisys-2022-0018>
12. A. Bouzidi, M. E. Riffi, Cat swarm optimization to solve job shop scheduling problem, 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), 202–205, (2014)
13. A. Bouzidi, M. E. Riffi, M. Barkatou, Cat swarm optimization for solving the open shop scheduling problem, *Journal of Industrial Engineering International*, 15, 367–378, (2019), <https://doi.org/10.1007/s40092-018-0297-z>
14. S. Mahmud, R. K. Chakraborty, A. Abbasi, M. J. Ryan, Switching strategy-based hybrid evolutionary algorithms for job shop scheduling problems, *Journal of Intelligent Manufacturing*, 33, 1939–1966, (2022), <https://doi.org/10.1007/s10845-022-01940-1>
15. A. Delgoshaei, M. K. Ariffin, Z. B. Leman, An Effective 4-Phased Framework for Scheduling Job-Shop Manufacturing Systems Using Weighted NSGA-II, *Mathematics*, **10**, 4607, (2022)
16. B. Khurshid, S. Maqsood, A hybrid evolution strategies-simulated annealing algorithm for job shop scheduling problems, *Engineering Applications of Artificial Intelligence*, 133, 108016, (2024)
17. J. F. Gonçalves, J. J. Magalhães Mendes, M. G. de, Resende, A hybrid genetic algorithm for the job shop scheduling problem, *European journal of operational research*, 167, 77–95, (2005)
18. J. E. Beasley, OR-Library: Distributing Test Problems by Electronic Mail, *Journal of the Operational Research Society*, 41, 1069–1072, (1990), <https://doi.org/10.1057/jors.1990.166>
19. H. Yu, Y. Gao, L. Wang, J. Meng, J, A hybrid particle swarm optimization algorithm enhanced with nonlinear inertial weight and Gaussian mutation for job shop scheduling problems, *Mathematics*, 8, 1355, (2020)