Proceedings of the
13th International Conference on Applied Informatics
Eger, Hungary, February 19–21, 2026
DOI: 10.17048/icai.2026.3
pp. 3–20

Combination of Ant Algorithms and Simulated Annealing in solving Flow Shop Scheduling Problem*

Anita Agárdi

University of Miskolc
anita.agardi@uni-miskolc.hu

Abstract. In this article, a common production scheduling task is presented and solved. This problem is the Flow Shop Scheduling. During the problem, given n jobs and m machines. All jobs must be processed in the same machine order. A job can only be started on a given machine if the processing on the previous machine has already been completed and the machine is free. The objective function of the problem is the minimization of the makespan. In this paper, two Hybrid Ant Algorithms were applied to the problem and solved the Taillard benchmark dataset. These two algorithms are the Hybrid Ant Colony System with Simulated Annealing and the Hybrid Rank Based Version of Ant System with Simulated Annealing.

Keywords: Ant Colony System, Rank Based Version of Ant System, Flow Shop Scheduling

2020 *Mathematics Subject Classification:* Primary 90B35; Secondary 90C59, 68W40, 90C27

1. Introduction

The efficient use of available resources is a key issue in the operation of modern industrial and service systems. During production, many tasks have to be performed on different machines, either in a specific order and possibly under a time limit. This falls under the task of production scheduling. The goal is usually to optimize

*Supported by the University Research Scholarship Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

some performance indicators. The performance indicator can be, for example, minimizing lead time, increasing machine utilization, reducing delays, minimizing the makespan.

Flow Shop Scheduling [19] is a common problem in production scheduling tasks, where n jobs and m machines are given. During the problem, all jobs must be processed in the same machine order. A job can only be started on a given machine if the processing on the previous machine has already been completed and the machine is free. The common objective function of the problem is to minimize the makespan. Over the years, many variations of the problem have been developed to suit real industry needs. Some of these specific constraints, types, and objective functions are as follows: In the case of Permutation Flow Shop Scheduling Problem (PFSP) [29] the jobs are processed in the same order on each machine. Non-permutation Flow Shop Scheduling Problem [26] is the case, when the order of jobs can vary from machine to machine. Blocking Flow Shop Scheduling Problem [30] is a task, when there is no buffer capacity between machines in the system. If the next machine is busy, the job on the previous machine will occupy that machine. No-wait Flow Shop Scheduling Problem [25] means that there can be no waiting between two consecutive operations. No-idle Flow Shop Scheduling Problem [24] is a case, when machines cannot stop between two consecutive jobs. In the case of Flexible Flow Shop Scheduling Problem [11], multiple parallel machines are available in one or more operation stages. Distributed Flow Shop Scheduling Problem [16] means, that the system contains multiple factories (production sites). The jobs have to be distributed among them. In the case of Stochastic Flow Shop Scheduling Problem [14] some parameters of the job are represented as random variables. Multi-objective Flow Shop Scheduling Problem [20] is a task, when multiple objective functions are optimized simultaneously.

In the following, some important publications are presented where the authors used the Ant Colony System algorithm to solve certain Flow Shop tasks.

Li Xiangyong, Baki Md Fazle and Aneja Yash P [13] investigated a Flow Shop Scheduling Problem with batching and the assumption of item availability with m machines. The objective of the problem is to minimize the total completion time. A single operator is required to process the jobs and set up the machines. At most one machine can be active in the system at a time. Each job needs to be processed only once. Data sets of different difficulty were used during testing. The sizes were as follows, where m is the number of machines and n is the number of jobs: $(m = 2, n = 5), (m = 2, n = 10), (m = 3, n = 10), (m = 5, n = 10)$ and $(m = 3, n = 20)$.

Chaouch Imen, Driss Olfa Belkahla and Ghedira Khaled [9] used a modified Ant Colony algorithm. They investigated a special version of the Flow Shop scheduling problem (Distributed Job Shop Scheduling Problem). In this model, jobs are distributed among different, geographically separated factories. The goal is to minimize the makespan. The following Ant Colony algorithms were used to solve the problem: ACO, Ant System (AS) and Modified Ant Colony Optimization (MACO). The model assumes that the tasks are independent of each other. Once a task is

assigned to a factory, it cannot be moved to another factory later. Interruption of operations is not allowed. A machine can only execute one operation at a time. The tests were performed on the well-known Taillard benchmark datasets.

Komaki GM, Sheikh Shaya and Malakooti Behnam [12] present a Flow Shop model that also includes assembly operations. The assembly Flow Shop system consists of a manufacturing or machining phase and then an assembly phase. The final products are assembled in a hierarchical structure. So, each component must first be machined separately. Then, the components must be assembled according to the predefined structure. The paper describes several different optimization methods, such as Artificial Bee Colony, Ant Colony Optimization, Genetic Algorithm, Particle Swarm Optimization, Simulated Annealing, Tabu Search, Variable Neighborhood Search. The objective function under study has several components: minimizing the makespan, reducing the total completion time, minimizing delays, and reducing inventory holding costs.

Chen Zhen, Zheng Xu, Zhou Shengchao, Liu Chuang and Chen Huaping [10] apply a Quantum-Inspired Ant Colony Optimization Algorithm (QIACO). The model uses two-phase permutation scheduling and batch processing machines in the Flow Shop task. During the operation of the algorithm, the ants are divided into two groups: one group prefers larger jobs, while the other selects smaller ones.

Ying Kuo-Ching and Liao Ching-Jong [28] use the Ant Colony System (ACS) algorithm to solve the Flow Shop Sequencing Problem. The results are also compared with other metaheuristic methods. The performance of the algorithm is tested on problems of different sizes during the experiments.

We can find many hybrid metaheuristic approaches in the literature, where the authors solve the classic Flow Shop problem or some variant of it. The article now highlights some publications, where the Taillard benchmark dataset is solved by the individual authors.

Zhou, Y., Chen, H., and Zhou, G. [31] solved the Taillard dataset with Invasive weed optimization algorithm. Although the results were better than the NEH algorithm, PSOvns and HDPSO, they were far from the best known value.

Wei, H., Li, S., Jiang, H., Hu, J., and Hu, J. [27] investigated the Hybrid Simulated Genetic Algorithm and tested on the Taillard benchmark dataset. The authors generated the initial values of the population not randomly, but with the MinMax (MM) and Nawaz–Enscore–Ham (NEH) algorithms. The results were also considerably distant from the best known value.

Tseng, L. Y., and Lin, Y. T. [23] developed and tested a hybrid Genetic Algorithm on the Taillard benchmark dataset. The GA algorithm was hybridized with two Local Search algorithms. The results were also not close to the best-known value.

Qu, C., Fu, Y., Yi, Z., and Tan, J. [17] tested the efficiency of the Flower Pollination Algorithm on the Taillard dataset. The authors also presented the results of hybridizing the Ant Colony Optimization algorithm with the Simulated Annealing algorithm, which were also far from the best known value.

The importance of Simulated Annealing-based improvement

In the previous work of the author, several metaheuristic methods have been implemented to solve Taillard benchmark problems. These algorithms were not applied in a hybrid form, but as standalone approaches.

Based on the results, it can be observed that these purely metaheuristic methods generally show significant deviations from the best known solutions of the benchmark datasets:

- Hill Climbing [3]
- Elitist Strategy of Ant System [3]
- Ant System [2]
- Rank Based Version of Ant System [4]
- MAX-MIN Ant System [1]

In this paper, a Simulated Annealing-based improvement approach is investigated, which was tested on several benchmark datasets. The algorithm achieved the best known value in many cases during each run. In those cases where deviations from the best known value were observed, the deviations remained small. Based on the relative standard deviation (RSD) values, it can be observed that the algorithm showed stable behavior, since the fitness values of the solutions obtained during different runs produced low standard deviations and, in many cases, nearly identical results. This suggests that the method is robust.

The paper is structured as follows. Section 2 presents the applied algorithms. Then Section 3 presents the test results. Section 4 provides a conclusion and future research direction.

2. Applied algorithms

This section presents the applied algorithms: the Ant Colony System algorithm, the Rank Based Version of Ant System algorithm, and Simulated Annealing.

2.1. Ant Colony System

The Ant Colony System [21] algorithm is a member of the Ant Colony Optimization (ACO) algorithm family. ACO algorithms are nature-inspired metaheuristic algorithms that maintain a population of solutions. They model the way ants search for food.

Constructing a route [21]: Initially, each ant starts from a randomly selected permutation element. At each step, the ants decide which permutation segment to follow with a given probability.

The probability that ant k, which is lastly selected permutation element i, will select element j in iteration t is given by:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \quad \text{when } j \in N_i^k$$

where $\eta_{ij} = \frac{1}{d_{ij}}$ is the reciprocal of the distance between two elements. $\tau_{ij}(t)$ is the current pheromone level of the pair (i, j). α and β are pheromone and distance control parameters. N_i^k denotes the set of elements not yet selected by ant k.

Global Pheromone Update [21]: During the algorithm, only the ant – which represents the globally best solution – places pheromone.

The update is done using the following formula:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \rho\,\Delta\tau_{ij}^{gb}(t)$$

where $\Delta\tau_{ij}^{gb}(t) = \frac{1}{L^{gb}}$, L^{gb} is the fitness of the globally best ant, ρ is the evaporation coefficient $(0 < \rho \leq 1)$ and $\tau_{ij}(t)$ is the current pheromone level of the pair (i, j). In this procedure, only the solution of best ant is updated.

Local pheromone update [21]: The algorithm also uses local updating:

$$\tau_{ij} = (1-\xi)\,\tau_{ij} + \xi\,\tau_{ij}^0$$

where $0 < \xi < 1$, τ_{ij}^0 is the initial pheromone level and τ_{ij} is the current pheromone value of the permutation pair (i, j).

This step reduces the pheromone level of the given edge, thus facilitating a wider search space.

Steps of the algorithm

The algorithm consists of the following steps [21]:

1. **Initialization of pheromone values.** This can be done based on an initial solution, or each edge can be given the same pheromone value.
2. **Route construction.**
   - Random selection of initial permutation element (solution element).
   - Calculating the selection probabilities of elements that have not yet been selected.
   - Select the next element.
3. **Local pheromone update.**

$$\tau_{ij} = (1-\xi)\,\tau_{ij} + \xi\,\tau_{ij}^0$$

4. Repeat the above steps until the ant has completed its route, i.e. the algorithm has produced a solution.

5. **Global pheromone update.** After all ants have created their solution, we perform an update on the best solution:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \rho\,\Delta\tau_{ij}^{gb}(t)$$

6. Steps 2–3 are repeated until the stopping condition is met. The stopping condition can be reaching a given number of iterations, convergence, or running time.

2.2. Rank Based Version of Ant System

The Rank Based Version of Ant System (RBVAS) [7] is an improved version of the Ant System algorithm. This algorithm is also a member of the Ant Colony Optimization family. Only the best ants contribute to increase in pheromone values. The algorithm ranks the ants in each iteration based on their fitness values:

$$L^1(t) \leq L^2(t) \leq \cdots \leq L^m(t)$$

The amount of deposited pheromone depends on the rank (r) of the ant. Only the best ($w-1$) ant of the current iteration deposits pheromone on the visited road sections. The ant that provides the globally best solution will deposit pheromone with weight w. The ant r deposits pheromone with the following weight:

$$\max\{0, w-r\}$$

The modified pheromone update is done with the following formula:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \sum_{r=1}^{w-1}(w-r)\,\Delta\tau_{ij}^{r}(t) + w\,\Delta\tau_{ij}^{gb}(t)$$

where $\Delta\tau_{ij}^{r}(t) = \frac{1}{L^r(t)}$, $\Delta\tau_{ij}^{gb}(t) = \frac{1}{L^{gb}}$, $\tau_{ij}(t)$ is the pheromone level of edge (i,j) in iteration t, ρ is the evaporation parameter.

Steps of the algorithm

The algorithm consists of the following steps [21]:

1. **Initialization of pheromone values.**

2. **Route construction.** Each ant constructs a solution in the following way:

   - The ant randomly selects a starting solution (permutation) element.
   - Then, it calculates the probability for each element (i,j), where i is the last element of the partial solution and j is not yet included.

The probability is given by:

$$p_{ij}^k = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha [\eta_{il}]^\beta} \quad \text{if } j \in N_i^k$$

where:

- $\eta_{ij} = \frac{1}{d_{ij}}$,
- $\tau_{ij}(t)$ is the current pheromone level of pair (i, j),
- t is the iteration number,
- N_i^k is the set of elements not yet visited.

The next permutation element is selected based on the calculated probabilities. The steps are repeated until a complete permutation (solution) is obtained.

3. **Pheromone update.** After all ants have created their solutions, the pheromone values are updated. The modification is based on the globally best ant and the best $(w-1)$ ants of the given iteration.

   The update formula is:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \sum_{r=1}^{w-1} (w-r)\,\Delta\tau_{ij}^r(t) + w\,\Delta\tau_{ij}^{gb}(t)$$

   where:

   - $\Delta\tau_{ij}^r(t) = \frac{1}{L^r(t)}$,
   - $\Delta\tau_{ij}^{gb}(t) = \frac{1}{L^{gb}}$,
   - ρ is the evaporation rate.

4. **Iteration.** Steps 2–3 are repeated until a stopping condition is met. This can be reaching a given number of iterations, convergence, or a time limit.

2.3. Simulated Annealing

Simulated Annealing [6] is inspired by the metallurgical heat treatment process. In the hardening of metals, the material is first heated to a high temperature and then slowly cooled. As the temperature is gradually reduced, the material adopts a lower-energy crystal structure.

During the search, random small modifications are taken to the current solution. That is, generating neighbors of the current solution. If the new solution is better than the current solution, it is accepted. If the new solution is worse, it can only be accepted with a certain probability. This acceptance probability depends on the energy change (ΔE) and decreases when decreasing temperature (T). In the initial stages of the algorithm, worse solutions are more likely to be accepted. Accepting worse solutions helps to escape from local optima.

Steps of the algorithm

The algorithm consists of the following steps [6]:

1. **Create an initial solution.** Select a possible solution, which will be the current solution: S_{current}.
2. **Create a neighboring solution.** Modify the current solution to produce a neighboring solution: S_{neighbor}. In this study, the 2-opt [15] operator was used to create the neighbor solution.
3. **Calculate the energy change.**

$$\Delta E = E(S_{\text{neighbor}}) - E(S_{\text{current}})$$

4. **Acceptance decision.** If $\Delta E < 0$, then the new solution is better, therefore:

$$S_{\text{current}} = S_{\text{neighbor}}$$

   Otherwise, the new solution is accepted only with the following probability:

$$P\left(e^{-\Delta E/T} > \text{rand}[0,1]\right)$$

5. **Internal iteration.** Steps 2–4 are repeated L times (process length).
6. **Temperature reduction.** This is done using the following formula:

$$T = \alpha \cdot T$$

7. **Outer loop.** Steps 2–6 are repeated until a stopping condition is met. The stopping condition can be a given number of iterations, running time, or convergence.

2.4. Improvement with Simulated Annealing

The presented algorithms were improved using the Simulated Annealing method. The improvement was created in a similar way as in the publication [5] with the difference that here the Simulated Annealing improvement was not performed as the last step of each iteration, but as an intermediate step of the algorithm-specific operators. In addition, an Iterative Simulated Annealing is also examined, when the algorithm simply re-rans several Simulated Annealings.

The improvement with Simulated Annealing (similar to the publication [5]) was performed in the following way for each iteration of the algorithm:

1. Generate and evaluate the initial population.
2. Iteratively perform the following operations until the stopping condition is met:

(a) Iterate over the elements of each population and perform the given algorithm-specific operators on each element.

(b) Iterate over the elements of each population again and improve each element using the Simulated Annealing method.

2.5. Problem representation

A permutation representation method is used for the task. The solution is described as a sequence of jobs (a permutation) $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. The permutation defines a common processing order of jobs on all machines.

Let $p_{k,g}$ denote the processing time of job k on machine g, where n is the number of jobs and m is the number of machines.

The completion time of the job at position t in the permutation on machine g, denoted by $C_{g,t}$, is computed recursively as follows:

$$C_{g,t} = \max(C_{g-1,t},\, C_{g,t-1}) + p_{\pi_t,g}$$

with boundary conditions:

$$C_{0,t} = 0 \quad \forall t = 1, \ldots, n, \qquad C_{g,0} = 0 \quad \forall g = 1, \ldots, m.$$

The term $C_{g-1,t}$ ensures that a job cannot start on machine g before it has completed processing on the previous machine. $C_{g,t-1}$ ensures that machine g processes jobs sequentially according to the permutation.

The objective function is makespan minimization (completion time of the last job on the last machine):

$$C_{\max} = C_{m,n}.$$

3. Test results

This section presents the test results of the Taillard [22] dataset, which is a Flow Shop Scheduling benchmark dataset. It is one of the most well-known test datasets for production scheduling problems. It was published by Éric Taillard in 1993. The Taillard examples contain problems of different sizes, consisting of different numbers of jobs and machines. For the Taillard benchmark dataset, the objective function is the makespan minimization. The tables consist of the following columns: the Taillard dataset number, the minimum, maximum and average fitness GAP values of the runs. The RSD values are also presented in the table.

The GAP value is calculated in the following way:

$$\mathrm{GAP}(\%) = \frac{C_{\mathrm{alg}} - C_{\mathrm{best}}}{C_{\mathrm{best}}} \cdot 100$$

where C_{alg} is the objective function value obtained by the algorithm, C_{best} is the value of the best known solution, GAP(%) is the relative deviation in percentage.

The RSD (Relative Standard Deviation) is calculated in the following manner:

$$RSD = \frac{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}}{\bar{x}} \cdot 100$$

3.1. Parameter settings and experimental environment

The algorithms were applied with the following parameter settings during the experiments.

Hybrid Ant Colony System (HACS):

- number of ants: 35,
- evaporation rate (ρ): 0.8,
- pheromone effect (α): 1,
- heuristic information effect (β): 2,
- local update parameter (κ): 0.8.

Hybrid Rank Based Version of Ant System (HRBVAS):

- number of ants: 35,
- evaporation rate (ρ): 0.8,
- pheromone effect (α): 1,
- heuristic information effect (β): 2,
- rank-based weighting parameter (w): 6.

Simulated Annealing (SA):

- number of iterations: 4000,
- initial temperature: 1000,
- cooling rate (α): 0.1.

In the case of Ant-based algorithms, the stopping condition was the convergence.

During the evaluation of the experimental results, 10 independent runs were performed for each test case. The algorithms were implemented in Java and run on a personal computer. For the smallest problems, the running time was 20-30 seconds, while for the largest instances, the running time was significantly longer, about 3-4 minutes.

3.2. Hybrid Ant Colony System test results

The Table 1 shows the performance of the Hybrid Ant Colony System (HACS) algorithm on the Taillard benchmark problem. The table includes the minimum, maximum, and average GAP values from the best known (BK) solutions in percentage form. It also contains the RSD values.

Table 1. Statistical results of Ta for Hybrid ACS.

Ta	Min	Max	Avg	RSD	Ta	Min	Max	Avg	RSD
1	matches best known				26	0.9883	1.4825	1.3208	0.1998
2	0.0000	0.0736	0.0074	0.0233	27	1.8038	2.3757	2.0326	0.2077
3	0.0000	0.3700	0.0370	0.1170	28	0.6818	2.0455	1.5636	0.5167
4	0.0000	0.3094	0.1160	0.1422	29	1.6987	2.4587	2.0563	0.3136
5	matches best known				30	1.2397	2.7089	2.2406	0.5936
6	matches best known				31	matches best known			
7	0.4052	0.9724	0.4619	0.1786	32	0.1411	0.4587	0.2893	0.1438
8	matches best known				33	0.0000	0.1145	0.0305	0.0497
9	matches best known				34	0.0727	0.0727	0.0727	0.0000
10	matches best known				35	matches best known			
11	0.7585	2.0228	1.5044	0.3280	36	0.0000	0.0707	0.0424	0.0387
12	0.7836	1.6275	1.1995	0.2685	37	0.0000	0.1101	0.0220	0.0492
13	1.4037	2.5401	1.7781	0.3423	38	0.0000	0.4100	0.2684	0.1974
14	0.5084	2.4691	1.4379	0.5675	39	0.1176	0.3527	0.2508	0.0941
15	0.4228	2.0437	1.0500	0.4803	40	matches best known			
16	0.3579	1.2885	0.7731	0.2818	41	2.8753	3.6108	3.3768	0.2801
17	0.3369	1.4151	0.7884	0.3057	42	2.6160	3.5926	3.2368	0.3642
18	0.7802	2.1456	1.6450	0.3932	43	2.8179	4.0859	3.6210	0.4620
19	0.5022	1.0672	0.8035	0.2232	44	0.6856	1.5344	1.1688	0.3073
20	0.0000	1.6970	0.9742	0.4991	45	2.4530	3.3266	3.0712	0.3501
21	1.6108	2.4380	1.9417	0.3656	46	2.0625	2.5615	2.4152	0.2087
22	1.7151	2.0962	1.9438	0.1852	47	1.6166	2.2308	2.0045	0.2435
23	1.9776	2.3646	2.1152	0.1440	48	0.5927	1.9098	1.3434	0.5424
24	1.1696	2.1592	1.6104	0.3602	49	1.6569	2.5889	1.9468	0.3790
25	1.2658	2.0079	1.6150	0.2665	50	2.4796	3.1974	2.8516	0.2887

Small-scale problems (Ta1–Ta10) For small problems, HACS performs very well. In several cases (Ta1, Ta5, Ta6, Ta8, Ta9, Ta10) the minimum, maximum and average values are the same. Here, the GAP is 0% in all cases. For the Ta2–Ta4 instances, the maximum GAP is also low. For the Ta7 dataset, the average GAP is 0.46%. This is still low. Here, Hybrid ACS is practically optimal or close to optimal, and the variance is also minimal.

Medium-sized problems (Ta11–Ta30) Larger GAP values can be observed in the case of medium-sized problems. The maximum GAP values are between 1.0-3.0%. For the Ta11–Ta20 instances, the average GAP is around 1%. For the Ta21–Ta30 problems, however, in the most cases the average GAP exceeds 1.5%. The maximum GAP values are below 2.0% in several cases.

Larger problems (Ta31–Ta50) For Ta31–Ta40 data, the GAP is again 0% or very low in most of the test runs. For Ta31, Ta35 and Ta40 the test results matches the best known values. For Ta41–Ta50 data, performance degradation is

observed. The average GAP values are mostly between 2–3.6%. E.g. Ta43: avg GAP ≈ 3.62%, Ta41: avg GAP ≈ 3.38%, Ta42: avg GAP ≈ 3.24%, Ta50: avg GAP ≈ 2.85%.

Stability test (min–max difference) For smaller problems, the min and max values are often the same, this means that the standard deviation is very low. For larger problems, the min–max difference increases (e.g. Ta41–Ta50).

3.3. Hybrid Rank Based Version of Ant System test results

The Table 2 indicates the performance of the Hybrid Rank Based Version of Ant System (HRBVAS) algorithm on the Taillard benchmark dataset. The table shows the minimum, maximum and average fitness values for each problem, as well as the deviations from the best known solution (GAP).

Table 2. Statistical results of Ta for Hybrid RBVAS.

Ta	Min	Max	Avg	RSD	Ta	Min	Max	Avg	RSD
1	matches best known				26	1.1231	1.4376	1.2579	0.1331
2	matches best known				27	1.5398	2.1557	1.9534	0.2414
3	0.0000	0.3700	0.0370	0.1170	28	0.9545	2.7727	1.9182	0.6919
4	0.0000	0.3094	0.1237	0.1596	29	1.9222	2.3692	2.1368	0.1943
5	0.0000	0.1619	0.0162	0.0512	30	1.6988	2.7089	2.1212	0.3950
6	matches best known				31	matches best known			
7	0.4052	0.4052	0.4052	0.0000	32	0.1411	0.4587	0.3035	0.1129
8	matches best known				33	0.0382	0.1145	0.0687	0.0418
9	matches best known				34	0.0727	0.0727	0.0727	0.0000
10	matches best known				35	matches best known			
11	0.6321	1.6435	1.1631	0.3400	36	0.0000	0.0707	0.0424	0.0387
12	0.3014	1.6878	1.1875	0.3820	37	0.0000	0.2569	0.1908	0.1080
13	0.8690	2.2727	1.5508	0.4043	38	0.0000	0.4100	0.2013	0.2048
14	0.6536	1.5251	1.1983	0.2954	39	0.0784	0.3527	0.2665	0.1153
15	0.4933	1.6913	1.0500	0.4190	40	matches best known			
16	0.0716	1.2885	0.7373	0.3593	41	3.0759	3.6108	3.3567	0.2225
17	0.4717	1.0108	0.7682	0.1761	42	2.8253	3.7321	3.3136	0.3500
18	0.7802	2.2757	1.5735	0.4255	43	3.3815	4.1916	3.6703	0.2956
19	0.5650	1.1299	0.9102	0.2180	44	1.1753	2.1221	1.6259	0.3334
20	0.8171	1.5085	1.1754	0.2192	45	2.9570	3.5282	3.1855	0.2225
21	1.4802	2.1332	1.8459	0.2971	46	1.9295	2.7944	2.3353	0.3752
22	1.2863	2.8109	1.8866	0.5808	47	1.8429	2.1662	2.0498	0.1374
23	1.2898	2.2786	1.7713	0.3783	48	1.5805	2.0086	1.8110	0.1696
24	1.4845	1.8444	1.6464	0.1583	49	1.8985	3.0376	2.4094	0.4898
25	1.4404	2.3134	1.8856	0.3512	50	2.5449	3.0669	2.8711	0.2139

Small-scale problems (Ta1–Ta10) In the case of Ta1-Ta10, the performance of the algorithm gave extremely favorable results. In several cases, the minimum, maximum and average values are the same. For most individuals, the algorithm resulted in a GAP of 0%. There was a smaller difference only in the case of Ta3, Ta4, Ta5 and Ta7. Here, the average GAP is between 0.016–0.4%.

Medium-sized problems (Ta11–Ta20) In this range, the GAP values are already higher. The average GAP is typically between 1–2%. The largest average

difference is observed in the case of Ta13 and Ta18. Here the difference is approximately 1.5%. The minimum GAP in most cases is between 0.3–0.8%.

Large-scale problems (Ta21–Ta30) For these data sets, the average GAP values are around 2% in most cases. In the case of Ta29 and Ta30, the average difference already exceeds 2%. The minimum GAP is already above 1% for several instances. In a larger search space, the result of the Hybrid RBVAS algorithm is therefore further away from the best known result.

Large-scale problems (Ta31–Ta40) Here again an improving trend can be observed. In several instances, the algorithm resulted in a minimum GAP value of 0% (e.g. Ta36, Ta37, Ta38). The average GAP values typically ranged between 0–0.4%. The stability of the algorithm is good, the standard deviation is moderate.

Largest size problems (Ta41–Ta50) Here again, a significant performance degradation is observed. The average GAP values were between 2–3.6% in the most cases. In the case of Ta43, the average deviation reached 3.67%. The maximum GAP exceeds 3% in several run cases regarding Ta41-Ta50 test instances.

Summary evaluation The performance of the Hybrid Rank Based Version of Ant System gave optimal or near-optimal results for small-scale tasks. Here, the GAP values were between 0-0.4% in most of the cases. For medium-sized tasks, the runs had a GAP value of around 1-2%. For Ta21-Ta30 individuals, the GAP was already 1.5-2.8% in most of the test runs, while for Ta31-Ta40 the algorithm is effective (0-0.45% GAP results).

3.4. Iterated Simulated Annealing algorithm results

The Table 3 presents the performance of the Iterated Simulated Annealing (ISA) algorithm on the Taillard benchmark problems. The table includes the minimum, maximum, and average fitness values, as well as their GAP values compared to the best known solutions.

Small-scale problems (Ta1–Ta10) For most instances (Ta1–Ta3, Ta5–Ta6, Ta8–Ta10) all runs reached the best known solution. In the case of Ta4 and Ta5, minimal deviations are observed. Here the average GAP is 0.10% and 0.4%. For small problems, the algorithm provides near-optimal solutions. The standard deviation here was low.

Medium-sized problems (Ta11–Ta30) In the case of medium-sized problems, some deterioration can already be observed. However, the difference was mostly around 1–2%. In the case of Ta11–Ta20, the average GAP values were between 0.7–1.7%. The maximum GAP in several cases exceeds 2%. The min–max differences are smaller, which means a relatively stable search. In the case of Ta21–Ta30, the

Table 3. Statistical results of Ta for ISA.

Ta	Min	Max	Avg	RSD	Ta	Min	Max	Avg	RSD
1	matches best known				26	0.8985	1.4376	1.2129	0.1935
2	matches best known				27	1.4078	2.3757	1.9358	0.3713
3	matches best known				28	1.3182	2.2727	1.9545	0.3622
4	0.0000	0.3867	0.1005	0.1631	29	1.0729	2.2798	1.8328	0.5240
5	matches best known				30	1.5611	3.3517	2.4793	0.7583
6	matches best known				31	matches best known			
7	0.4052	0.4052	0.4052	0.0000	32	0.1411	0.4940	0.2541	0.1438
8	matches best known				33	0.0000	0.1145	0.0458	0.0418
9	matches best known				34	0.0727	0.0727	0.0727	0.0000
10	matches best known				35	matches best known			
11	0.4425	2.0228	1.4349	0.4513	36	0.0000	0.0707	0.0566	0.0316
12	1.0247	1.7480	1.3804	0.2179	37	0.0000	0.2569	0.0514	0.1148
13	0.8690	2.2727	1.7380	0.3321	38	0.1118	0.4100	0.2758	0.1330
14	0.5084	1.5977	1.2128	0.3945	39	0.1176	0.3527	0.2351	0.1173
15	0.3524	2.0437	1.1557	0.5246	40	matches best known			
16	0.2863	1.5032	0.8661	0.3850	41	2.6747	3.3768	3.0425	0.2482
17	0.1348	1.3477	0.7480	0.4776	42	2.3369	3.7670	3.0206	0.5431
18	1.2354	1.8856	1.5670	0.2327	43	2.7827	4.0155	3.5717	0.4825
19	0.1255	1.1927	0.8537	0.3374	44	1.2733	1.6650	1.4757	0.1481
20	0.4400	1.6342	1.1691	0.4444	45	2.8562	3.5618	3.1519	0.3236
21	1.2625	2.2638	1.6282	0.3815	46	2.2289	2.7611	2.5815	0.2059
22	1.2863	2.3821	1.7723	0.4019	47	1.7459	2.1662	1.9657	0.1868
23	1.0318	2.4076	1.7885	0.5012	48	1.0207	1.8110	1.5081	0.2914
24	1.5744	2.4741	1.8893	0.3532	49	2.2437	3.0031	2.6372	0.3292
25	1.7896	2.0952	1.9380	0.1502	50	2.4144	2.9690	2.6884	0.2042

average GAP values were typically between 1.1–2.4%. The largest difference was in the case of Ta30, here the average GAP was 2.4%, the maximum GAP was 3.3%.

Large-scale problems (Ta31–Ta50) For the Ta31–Ta40 datasets, the algorithm gave 0% or very low deviations from the best known solution in several cases. The average GAP values were typically between 0–0.28%. Stability is favorable in this range. The min–max differences are small. A significant performance decrease can be observed for the Ta41–Ta50 datasets. The average GAP values typically resulted in values between 1.4–3.5%. The largest average deviation was in the case of Ta43, where the average GAP was 3.57%. For several instances, such as Ta41, Ta42, Ta43, Ta45 the average deviation exceeds 3%. For larger problems, ISA shows a significant difference compared to the best known solutions. Here, the standard deviation also increases significantly.

Stability testing For small data sets, the minimum and maximum GAP values are often the same or the difference between them is very small. Here, the standard deviation is very low. For medium problems, moderate standard deviation is observed. For large problems, the difference between the minimum and maximum GAP increases.

Summary evaluation The algorithm provided stable, near-optimal solutions on small-scale problems. For medium-scale problems, the average deviation was

between 1–2%. For large-scale problems, the deviation from the best solution was between 2–3.6%, and the standard deviation was much larger.

3.5. J. Carlier and C.R. Reeves benchmarks

Test runs were performed not only on the Taillard data set, but also on the J. Carlier (car) [8] and C.R. Reeve (rec) [18] data sets. This can be seen in Table 4.

Table 4. Car and Rec instances (HACS, HRBVAS, ISA comparison).

Alg	Inst	Min	Max	Avg	RSD	Alg	Inst	Min	Max	Avg	RSD
HACS	car1	matches best known				ISA	car1	matches best known			
HACS	car2	matches best known				ISA	car2	matches best known			
HACS	car3	matches best known				ISA	car3	matches best known			
HACS	car4	matches best known				ISA	car4	matches best known			
HACS	rec1	0.1604	0.1604	0.1604	0.0000	ISA	rec1	0.0000	0.1604	0.1283	0.0676
HACS	rec3	matches best known				ISA	rec3	matches best known			
HACS	rec13	1.2435	2.0725	1.5492	0.2779	ISA	rec13	1.0363	2.0725	1.5492	0.3222
HACS	rec19	1.3856	2.2458	1.9546	0.2736	ISA	rec19	1.3850	2.2940	1.9160	0.2746
HRBVAS	car1	matches best known									
HRBVAS	car2	matches best known									
HRBVAS	car3	matches best known									
HRBVAS	car4	matches best known									
HRBVAS	rec1	0.1604	0.1604	0.1604	0.0000						
HRBVAS	rec3	matches best known									
HRBVAS	rec13	0.8808	1.8135	1.3980	0.3196						
HRBVAS	rec19	1.4335	2.2937	1.8968	0.2720						

The table also shows that in many case the algorithms reached the best known value during each run. Although they did not reach the best known value during the most complex test data, but they were close to it.

3.6. Summary of the test results

This section presents a comparison of the Hybrid Ant Colony System, the Hybrid Rank Based Version of Ant System and the Iterated Simulated Annealing algorithms. First, the test results are summarized in a table. The Table 5 contains the following columns: Taillard benchmark dataset identifier, the best algorithm and the minimum fitness value of the best algorithm.

When examining the performance of the three metaheuristic algorithms on the Taillard benchmark dataset, it can be observed that the algorithms were able to achieve the best known solution in several cases. On the first ten datasets (except Ta07), all three methods achieved the best known value. For larger problems, the Hybrid Ant Colony System (HACS) gave better results on several datasets. The HACS algorithm gave the best result in 13 times, the Hybrid Rank Based Version of Ant System in 7 times, and the Iterated Simulated Annealing in 14 times. In addition, the algorithms provided identical results in 17 times.

Table 5. Summary of best algorithm and minimum values for Ta instances.

Ta	Best	Min	Ta	Best	Min
Ta1	same	1278	Ta26	ISA	2246
Ta2	same	1359	Ta27	ISA	2305
Ta3	same	1081	Ta28	HACS	2215
Ta4	same	1293	Ta29	ISA	2261
Ta5	same	1235	Ta30	HACS	2205
Ta6	same	1195	Ta31	same	2724
Ta7	same	1239	Ta32	same	2838
Ta8	same	1206	Ta33	HACS=ISA	2621
Ta9	same	1230	Ta34	same	2753
Ta10	same	1108	Ta35	same	2863
Ta11	ISA	1589	Ta36	same	2829
Ta12	HRBVAS	1664	Ta37	same	2725
Ta13	HRBVAS = ISA	1509	Ta38	HACS=HRBVAS	2683
Ta14	HACS=ISA	1384	Ta39	HRBVAS	2554
Ta15	ISA	1424	Ta40	same	2782
Ta16	HRBVAS	1398	Ta41	ISA	3071
Ta17	ISA	1486	Ta42	ISA	2934
Ta18	HACS=HRBVAS	1550	Ta43	ISA	2918
Ta19	ISA	1595	Ta44	HACS	3084
Ta20	HACS	1591	Ta45	HACS	3049
Ta21	ISA	2326	Ta46	HRBVAS	3064
Ta22	HRBVAS=ISA	2126	Ta47	HACS	3143
Ta23	ISA	2350	Ta48	HACS	3055
Ta24	HACS	2249	Ta49	HACS	2945
Ta25	HACS	2320	Ta50	ISA	3139

4. Summary

In this article, a well-known production scheduling problem, the Flow Shop Scheduling, was investigated. The problem was solved with three algorithms, namely Hybrid Ant Colony System, Hybrid Rank Based Version of Ant System and Iterated Simulated Annealing. Hybrid ACS and Hybrid RBVAS mean that the algorithm also applies an improvement with the Simulated Annealing algorithm. After the introduction of the Flow Shop task and the most important publications on the topic, the article presented the applied algorithms. This was followed by the presentation and evaluation of the test results. The summary showed that the Hybrid Ant Colony System algorithm gave the best result in 13 times on the Taillard dataset, Hybrid Rank Based Version of Ant System in 7 times, and Iterated Simulated Annealing in 14 times. In addition, the algorithms gave the same results in 17 times. The effective hybridization of other metaheuristic algorithms with the help of Simulated Annealing could be a further research topic. In addition, testing other optimization problems, such as the Traveling Salesman Problem or the Vehicle Routing Problem, as well-known logistics discrete optimization problems, may also become a research focus.

References

[1] A. Agárdi: *Application of the Max-Min Ant System on Flow Shop Scheduling Problems*, Production Systems and Information Engineering 13.1 (2025), pp. 1–13.

[2] A. Agárdi: *Efficiency analysis of the Ant System algorithm on the Flow Shop Scheduling Problem*, Production Systems and Information Engineering 12.1 (2024), pp. 29–40.

[3] A. Agárdi: *Efficiency analysis of the Hill Climbing and Elitist Strategy of Ant System in the application of Flow Shop Scheduling Problems*, Production Systems and Information Engineering 13.2 (2025), pp. 1–17.

[4] A. Agárdi: *Rank-Based Version of Ant System in production scheduling*, Production Systems and Information Engineering 13.1 (2025), pp. 14–25.

[5] A. Agárdi, K. Nehéz, O. Hornyák, L. T. Kóczy: *A hybrid discrete bacterial memetic algorithm with simulated annealing for optimization of the flow shop scheduling problem*, Symmetry 13.7 (2021), p. 1131.

[6] D. Bertsimas, J. Tsitsiklis: *Simulated annealing*, Statistical science 8.1 (1993), pp. 10–15.

[7] B. Bullnheimer, R. Hartl, C. Strauss: *A new rank based version of the ant system: A computational study* (1997).

[8] J. Carlier: *Ordonnancements a contraintes disjonctives*, RAIRO-Operations Research 12.4 (1978), pp. 333–350.

[9] I. Chaouch, O. B. Driss, K. Ghedira: *A modified ant colony optimization algorithm for the distributed job shop scheduling problem*, Procedia Computer Science 112 (2017), pp. 296–305.

[10] Z. Chen, X. Zheng, S. Zhou, C. Liu, H. Chen: *Quantum-inspired ant colony optimisation algorithm for a two-stage permutation flow shop with batch processing machines*, International Journal of Production Research 58.19 (2020), pp. 5945–5963.

[11] S. Dauzère-Pérès, J. Ding, L. Shen, K. Tamssaouet: *The flexible job shop scheduling problem: A review*, European Journal of Operational Research 314.2 (2024), pp. 409–432.

[12] G. Komaki, S. Sheikh, B. Malakooti: *Flow shop scheduling problems with assembly operations: a review and new trends*, International Journal of Production Research 57.10 (2019), pp. 2926–2955.

[13] X. Li, M. F. Baki, Y. P. Aneja: *Flow shop scheduling to minimize the total completion time with a permanently present operator: Models and ant colony optimization metaheuristic*, Computers & Operations Research 38.1 (2011), pp. 152–164.

[14] C.-L. Liu, T.-H. Huang: *Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning*, IEEE transactions on systems, man, and cybernetics: systems 53.11 (2023), pp. 6836–6848.

[15] H. Okano, S. Misono, K. Iwano: *New TSP construction heuristics and their relationships to the 2-opt*, Journal of Heuristics 5.1 (1999), pp. 71–88.

[16] Y. Pan, K. Gao, Z. Li, N. Wu: *Solving biobjective distributed flow-shop scheduling problems with lot-streaming using an improved Jaya algorithm*, IEEE transactions on cybernetics 53.6 (2022), pp. 3818–3828.

[17] C. Qu, Y. Fu, Z. Yi, J. Tan: *Solutions to No-Wait Flow Shop Scheduling Problem Using the Flower Pollination Algorithm Based on the Hormone Modulation Mechanism*, Complexity 2018.1 (2018), p. 1973604.

[18] C. R. Reeves: *A genetic algorithm for flowshop sequencing*, Computers & operations research 22.1 (1995), pp. 5–13.

[19] D. A. Rossit, F. Tohmé, M. Frutos: *The non-permutation flow-shop scheduling problem: a literature review*, Omega 77 (2018), pp. 143–153.

[20] S. Shi, H. Xiong: *Solving the multi-objective job shop scheduling problems with overtime consideration by an enhanced NSGA-II*, Computers & Industrial Engineering 190 (2024), p. 110001.

[21] T. Stützle, M. Dorigo, et al.: *ACO algorithms for the traveling salesman problem*, Evolutionary algorithms in engineering and computer science 4 (1999), pp. 163–183.

[22] E. Taillard: *Benchmarks for basic scheduling problems*, european journal of operational research 64.2 (1993), pp. 278–285.

[23] L.-Y. Tseng, Y.-T. Lin: *A hybrid genetic algorithm for no-wait flowshop scheduling problem*, International journal of production economics 128.1 (2010), pp. 144–152.

[24] D. M. Utama, C. N. Al Imron: *A systematic literature review on no-idle flow shop scheduling problem*, in: Operations Research Forum, vol. 5, 1, Springer, 2024, p. 18.

[25] D. M. Utama, S. Z. Umamy, C. N. Al-Imron: *No-Wait Flow Shop scheduling problem: a systematic literature review and bibliometric analysis*, RAIRO-Operations Research 58.2 (2024), pp. 1281–1313.

[26] Z. Wang, B. Cai, J. Li, D. Yang, Y. Zhao, H. Xie: *Solving non-permutation flow-shop scheduling problem via a novel deep reinforcement learning approach*, Computers & Operations Research 151 (2023), p. 106095.

[27] H. Wei, S. Li, H. Jiang, J. Hu, J. Hu: *Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion*, Applied Sciences 8.12 (2018), p. 2621.

[28] K.-C. Ying, C.-J. Liao: *An ant colony system for permutation flow-shop sequencing*, Computers & Operations Research 31.5 (2004), pp. 791–801.

[29] A. N. H. Zaied, M. M. Ismail, S. S. Mohamed: *Permutation flow shop scheduling problem with makespan criterion: literature review*, J. Theor. Appl. Inf. Technol 99.4 (2021), pp. 830–848.

[30] F. Zhao, S. Di, L. Wang: *A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem*, IEEE Transactions on Cybernetics 53.5 (2022), pp. 3337–3350.

[31] Y. Zhou, H. Chen, G. Zhou: *Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem*, Neurocomputing 137 (2014), pp. 285–292.