

Keywords: swarm intelligence, snake optimiser, scheduling problem

Hassan ALMAZINI ¹, Salah MORTADA ^{2*}, Hussein Fouad ALMAZINI ³

¹ Southern Technical University, Iraq, hassan.f.abbas@sa-uc.edu.iq

² Basra University for Oil and Gas, Iraq, salah.mortada@buog.edu.iq

³ Shatt Al-Arab University College, Iraq, hussein.f.abbas@sa-uc.edu.iq

* Corresponding author: salah.mortada@buog.edu.iq

Modified snake optimiser algorithm for solving the permutation flow shop scheduling problem

Abstract

Minimising the makespan for large job sets is the goal of the algorithmically intensive NP-hard permutation flow shop scheduling problem (PFSP), which is crucial for real-world applications. Originally designed for continuous optimisation, the snake optimiser is a recently discovered swarm-based metaheuristic that may jeopardise its exploration-exploitation equilibrium in discrete domains, leading to premature convergence. To address this challenge, this work presents a modified snake optimiser (MSO) for (PFSP). To maintain diversity, prevent stagnation, and enhance solution quality, MSO uses three mutation strategies: swap, insertion, and inversion. 120 benchmark instances with job sizes ranging from 20 to 500 are used for extensive research. The results show that MSO performs comparably to well-known algorithms reported in the literature. MSO consistently outperforms its competitors on complex issues, demonstrating its usefulness as a robust approach for PFSP. Deviations from the optimal solutions range from 0.1% to 0.5%, and MSO consistently performs superior.

1. INTRODUCTION

Numerous optimisation techniques have been proposed in the field of artificial intelligence to identify significant patterns or resolve challenging problems considered hardest in NP (Nondeterministic Polynomial time). Several real-world problems have been observed in this context. These include: data mining (Jabbar & Ku-Mahamud, 2021), travelling salesman (Almazini et al., 2022), school timetabling (Arbaoui et al., 2022), quadratic assignment (Mzili et al., 2023), scheduling (Amin et al., 2025), and feature selection (Mortada, 2026). Since its introduction by Johnson (1954), the (PFSP) has been a focal point of scheduling research and has attracted a great deal of interest. This problem has been investigated through various theoretical, algorithmic and computational studies. The fundamental principle of PFSP is that n jobs must be executed on a series of m machines, with each job requiring processing on all machines in a consistent sequence. Johnson (1954) found that minimising the makespan in PFSP is NP-hard. Exact and approximate approaches (including heuristic and metaheuristic algorithms) demand excessive computational time when applied to large-scale problem instances.

The importance of PFSP across different industries has recently piqued researchers' interest, as it is a critical NP-hard problem with direct applications in manufacturing optimisation. These industries include logistics, communication, computer design and transportation. A wide range of optimisation algorithms has been employed in the literature to minimise the makespan of the optimal schedule and reduce computational complexity. Optimisation algorithms can develop and deploy optimum solutions for specific situations (Khalilpourazari & Khalilpourazary, 2018); however, exact optimisation algorithms struggle to address complicated problems. Therefore, metaheuristic techniques are typically employed to address PFSP problems, as they aim to discover the best solution in the shortest search time (Liu & Liu, 2013). Metaheuristic algorithms have been developed recently, and one of these algorithms is the snake optimiser (SO) (Hashim & Hussien, 2022). The SO is a new swarm intelligence algorithm inspired by how snakes mate. In SO, the population is divided into males and females, which interact differently depending on environmental conditions. Temperature and food quantity are the primary factors governing the search process, and each group is assessed independently. When no food is available, the algorithm enters the exploration phase, in which snakes spread

across the search space to discover promising new areas. Conversely, when food is available, the algorithm switches to the exploitation phase, where snakes converge on the best solutions, with males competing for mates and females adjusting their movements accordingly.

The SO algorithm's strong performance in resolving a diverse set of problems (AL-Attabi, 2022; Al-Shourbaji et al., 2022; Gogos, 2023; Lu et al., 2024; Ni et al., 2025; Zheng et al., 2025) motivates us to use it for PFSP. However, when SO is applied to discrete spaces, some limitations arise. For example, the delicate balance between exploration and exploitation can be disrupted, potentially leading to early convergence or entrapment in local optima. This study explores the potential of using SO to address PFSP. Mutation strategies are introduced to improve exploration and exploitation balance, strengthening exploration in the early steps whilst maintaining exploitation in the subsequent iterations to improve SO's ability to escape local optima. These mutation strategies, which improve exploration and exploitation, are the main advantages of the proposed MSO algorithm. They reduce the risk of early convergence by encouraging greater variance, which in turn increases the likelihood of finding optimal solutions. The main purpose of the PFSP is to determine the makespan, which is the shortest possible, thereby enabling MSO to achieve significant results in improving the job sequence.

2. RELATED RESEARCH

Several population-based algorithms have been employed to resolve PFSP. Tasgetiren et al. (2004) proposed an adaptation of particle swarm optimisation (PSO) for PFSP by combining an embedded local search to optimise solutions with a modified velocity operator and a discrete encoding method suitable for job permutations. On common benchmark datasets, their hybrid PSO outperforms conventional strategies and achieves lower makespan values. The research did point out a significant drawback, though: the algorithm sometimes experiences premature convergence due to decreased swarm diversity, which limits its capacity for exploration. For PFSP, Blum and Sampels (2002) proposed an ant colony optimisation-based method that uses heuristic data and pheromone trails to direct the creation of task variants. The method yields competitive results and performs well in small to medium-sized instances. The research did, however, identify limits in sensitivity to parameter adjustments and a progressive loss of exploratory capability as the number of iterations increases. These limitations cause the algorithm to become prematurely stagnant by reducing its ability to handle large or complex situations. The algorithm's ability to consistently minimise the makespan in difficult scheduling circumstances is further hampered by reduced exploration, which limits its ability to escape local optima.

To address the PFSP, Ruiz et al. (2006) devised a robust genetic algorithm to minimise makespan. To preserve diversity and enhance search efficiency, their approach utilises a range of crossover and mutation operators, together with task permutation representation. The strategy demonstrated competitive performance relative to Taillard benchmark examples when evaluated alongside other optimisation methods, especially for small and medium-sized issues. The limited scalability in large-scale issue cases, where maintaining high-quality solutions requires considerable processing effort, is a significant drawback. The compromise between solution precision and runtime constraints the algorithm's practical use, limiting its ability to consistently reduce makespan in large-scale scheduling problems. By transforming continuous position updates into discrete job exchanges and adding a local search strategy to improve solution accuracy, the researchers Fong et al. (2014) modified the firefly algorithm for PFSP. Compared to conventional methods, their method offers competitive solution quality, especially for medium-sized instances, and demonstrates efficacy in reducing span. Nonetheless, a notable limitation of the algorithm is its susceptibility to premature convergence. That is, population diversity tends to decrease after several iterations, causing the search to stagnate in local optima. This reduction in exploration capability hinders the algorithm's ability to consistently identify effective schedules, thereby undermining its ability to achieve the minimum makespan for complex or large-scale scheduling problems. Liang et al. (2022) offered a hybrid genetic algorithm (GA) for PFSP that includes a niche mechanism to maintain population diversity, three specialised crossover operators to improve recombination, and the Nawaz–Enscore–Ham (NEH) heuristic for initialisation. Experimental results on the Taillard benchmark sets show that this hybrid technique yields very competitive makespan values and proves more resilience than conventional genetic algorithms. Despite these advantages, the research identified a significant constraint on parameter sensitivity, namely the correlation between crossover mix and niche radius, which requires meticulous calibration across different issue sizes. This dependency ultimately affects the algorithm's ability to consistently reduce makespan across various PFSP scales, diminishing its portability and

potentially reducing its effectiveness across diverse instances. In a discrete framework for permutations, Huang et al. (2023) offered WPO, which integrates the whale optimisation algorithm (WOA; global encircling/search) with particle swarm optimisation (PSO; velocity-based refinement). Compared to WOA, PSO, and other contemporary hybrids, WPO consistently produces better or equal makespan values over the whole Taillard 20–500 task suite, especially for the most difficult 200–500 instances. The algorithm's sensitivity to the balance between whale and particle phases is a major problem; incorrect choices may reduce diversity and affect the minimum makespan.

The adaptive dynamic neighbourhood crow search algorithm (AdnCSA) (Zhao et al., 2024) is a discrete variant of the crow search (CS) method for PFSP. It handles job permutations using an adaptive neighbourhood-size method throughout the search, NEH-based seeding, and smallest-position-value encoding. When evaluated on Taillard benchmark cases with medium-to-large problem sizes (100-500 tasks), the method outperformed the baseline CS, GA, and PSO variants in terms of mean and best makespan values. The algorithm's reliance on the neighbourhood adaptation schedule, nevertheless, is a major disadvantage; if the neighbourhood size is not properly updated, the search may halt, limiting exploration. This sensitivity may affect the algorithm's ability to achieve the shortest makespan, making it more difficult to consistently find optimal or near-optimal sequences. An improved salp swarm algorithm (ISSA) was presented by Cai and Chen (2025) to address the integrated permutation flow shop vehicle routing issue. To minimise overall costs, including production, transportation, and delays, ISSA integrates distribution routing and production scheduling. ISSA performed competitively in the PFSP component compared with traditional metaheuristics, often approaching the best-known solutions. However, since its multi-objective design balances makespan with routing and delivery costs, the algorithm's primary drawback in the context of PFSP is that it is not specifically designed to minimise makespan alone. Furthermore, in many instances, the research did not compare ISSA with the most popular specialised PFSP algorithms. Therefore, although ISSA is stronger than simulated annealing (SA), GA, and PSO, its performance relative to leading specialised PFSP methods remains unclear. A summary of previous studies reveals that some showed early convergence in solutions, attributed to a lack of population diversity among solutions. Furthermore, many of the results in these studies were not robust compared to the best-known solutions. The findings of this study contribute to enriching the literature related to the MSO algorithm by developing an adaptive mechanism to integrate three modification operators, namely swap, insertion, and inversion, and employing them selectively within the different stages of the optimisation process. By achieving the optimal result of determining the minimal makespan, the primary goal of PFSP, the suggested MSO seeks to mitigate the aforementioned problems.

3. DESCRIPTION AND MATHEMATICAL FORMULATION OF PFSP

PFSP uses the permutation of n jobs or $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ to represent solutions. Every job consists of m processes, each implemented by a separate machine. All jobs have zero release times, and once they are started, they cannot be stopped (pre-empted) by another job on the same system. PFSP aims to ascertain the best job permutation $\pi^* = (\pi_1^*, \pi_2^*, \dots, \pi_n^*)$ for processing on each machine whilst adhering to the makespan constraint and the set processing time $t_{j,k}$ for task j on machine k . Let $C(\pi_j, m)$ represent the time it takes for job π_j on machine m to finish. Subsequently, given job permutation π and the time required for the n job to complete, the following shows the computation for the m -machine problem.

$$\begin{aligned}
C(\pi_1, 1) &= t_{\pi_1, 1} \\
C(\pi_j, 1) &= C(\pi_{j-1}, 1) + t_{\pi_j, 1} \quad j = 2, \dots, n \\
C(\pi_1, k) &= C(\pi_1, k-1) + t_{\pi_1, k} \quad k = 2, \dots, m \\
C(\pi_j, k) &= \max(C(\pi_j - 1, k) + C(\pi_j, K-1)) \\
&\quad + t_{\pi_j, k} \quad j = 2, \dots, n; \quad k \\
&\quad = 2, \dots, m
\end{aligned} \tag{1}$$

The completion time of last job π_n on final machine m may be used to explicitly determine the makespan of permutation $C_{max}(\pi) = \pi_n, m$. Consequently, PFSP with the makespan criteria aims to identify the best permutation π^* amongst all the permutations Π , such that for any permutation π that belongs to Π , $C_{max}(\pi^*) \leq C(\pi_n, m)$.

4. METHODS

This section addresses the modification of the balance between exploration (global searching) and exploitation (local searching). The key contribution of the proposed MSO algorithm lies in the adaptive integration of swap, insertion, and inversion operators within the SO. Only one operator is selected and activated at each stage based on threshold values related to the algorithm's behaviour, and it is then employed selectively during the exploration and exploitation phases. This adaptive approach helps to enhance search diversity, reduce early convergence, and improve the quality of solutions to the PFSP problem. Mutation strategies are used in MSO to solve PFSP because they play a vital role in enhancing the search process. In this work, mutation strategies are integrated into the algorithm to improve its performance. This operation incorporates controlled randomisation, which enhances population diversity, prevents premature convergence to local optima and allows the algorithm to avoid stagnation. Three widely used mutation strategies are implemented: switch mutation (swapping two randomly selected tasks), insert mutation (moving a task to a randomly selected location), and reversal mutation (reversing the order of a randomly selected subsequence). Mutation improves the balance between exploration and exploitation by providing random variations. This balance enhances exploration in the early stages while maintaining the power of exploitation in later iterations. This increases the probability of achieving outstanding work sequences that reduce the time required to complete all jobs. Accordingly, this combination improves the scheduling optimisation algorithm's overall search capabilities while maintaining consistent performance across various functional task-scheduling test scenarios. This makes it useful for handling complex scheduling optimisation search environments. The development methods are explained below.

Swap mutation: Two jobs are randomly selected within the sequence and then swapped.

Example: sequence = [1, 3, 5, 2, 4]

Swap jobs at positions 2 and 4 $\rightarrow$ [1, 2, 5, 3, 4].

Insertion mutation: It selects one job from the sequence and inserts it into a different position, shifting the others.

Example: sequence = [1, 3, 5, 2, 4]

Remove job 5 at position 3 and insert it at position 5 $\rightarrow$ [1, 3, 2, 4, 5].

Inversion mutation: It randomly selects two cut points in the sequence and reverses the order of jobs between them.

Example: sequence = [1, 3, 5, 2, 4]

Select cut points between jobs 2 and 4 $\rightarrow$ reverse subsequence [3, 5, 2] $\rightarrow$ [1, 2, 5, 3, 4].

4.1. Research methodology

The research methodology, as shown in Figure 1, begins with defining the problem and objectives, followed by the design and implementation of the algorithm. The algorithm is then tested on standard problems, and its results are compared with other algorithms to analyse the performance of the MSO.

5. PROPOSED MSO ALGORITHM FOR PFSP

The proposed MSO for solving PFSP begins with population initialisation, where each candidate solution is encoded as a job permutation. The swarm is thereafter partitioned into two equal factions representing males and females, and the search is guided by temperature and food quantity. Each group is assessed based on the makespan performance. During the exploration phase (no food), solutions explore the search space broadly to avoid premature convergence. In the exploitation phase (food exists), the algorithm intensifies the search around promising regions to refine job sequences. Then, termination criteria, such as reaching the maximum iteration limit or achieving an acceptable solution quality, are evaluated to complete the optimisation process. There are six stages in the MSO algorithm.

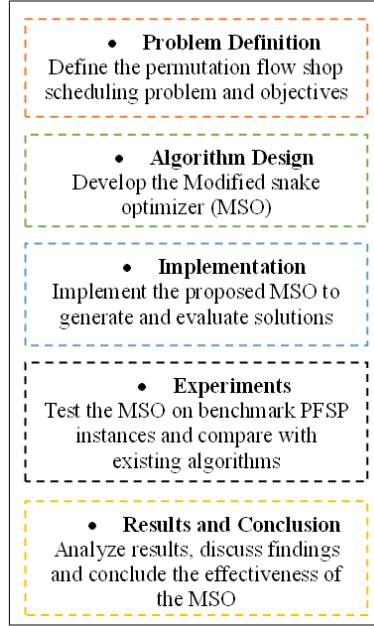


Fig. 1. illustrates the research methodology

Initialisation: The population is initialised by producing a set of random permutations of occupations. Every permutation denotes a potential order in which n workloads are distributed across m machines. By randomly rearranging the task indices so that each member of the population belongs to a distinct work ordering, many candidate solutions are generated. Ensuring variety in the search space is the goal. This randomness helps the algorithm avoid bias at the start and enables broad exploration of potential schedules, thereby increasing the chances of finding high-quality solutions as the optimisation progresses in PFSP.

Splitting the swarm evenly into male and female groups: In MSO, the population consists of 50% males and 50% females, and it is split into two groups by using Eqs. (1) and (2).

$$N_M \approx N/2 \quad (1)$$

$$N_f = N - N_M \quad (2)$$

where N denotes the total number of individuals, N_M represents the count of male individuals and N_f signifies the count of female individuals.

Group assessment, including the determination of food quantity and temperature: The best male ($f_{best,m}$), best female ($f_{best,f}$) and food location (f_{food}) are obtained. Then, the finest member is chosen from each category. Food quantity (Q) may be determined using the Eq. (3).

$$Q = c_1 * \exp\left(\frac{t-T}{T}\right) \quad (3)$$

where c_1 is a constant value that is equivalent to 0.5

The temperature, denoted as Temp, may be expressed using Eq. (4).

$$Temp = \exp\left(\frac{-t}{T}\right) \quad (4)$$

where t is the iteration that is currently being performed, and T is the maximum iteration count.

Exploration task: When Q is below the threshold of 0.25, the snakes search for food by randomly selecting a position and modifying their location via swap mutation.

Exploitation task: When Q and Temp exceed their respective thresholds (0.6 for Temp), the snakes go to the best individuals. If the temperature is below 0.6, the snakes will use an insertion mutation to fight or an inversion mutation to mate.

Checking terminating conditions: The terminating conditions are defined to control when the search process should stop. The algorithm continues this process until the maximum number of iterations is reached. Then, the procedure is completed, and the best solution found is recorded. Figure 2 illustrates the MSO algorithm's flowchart.

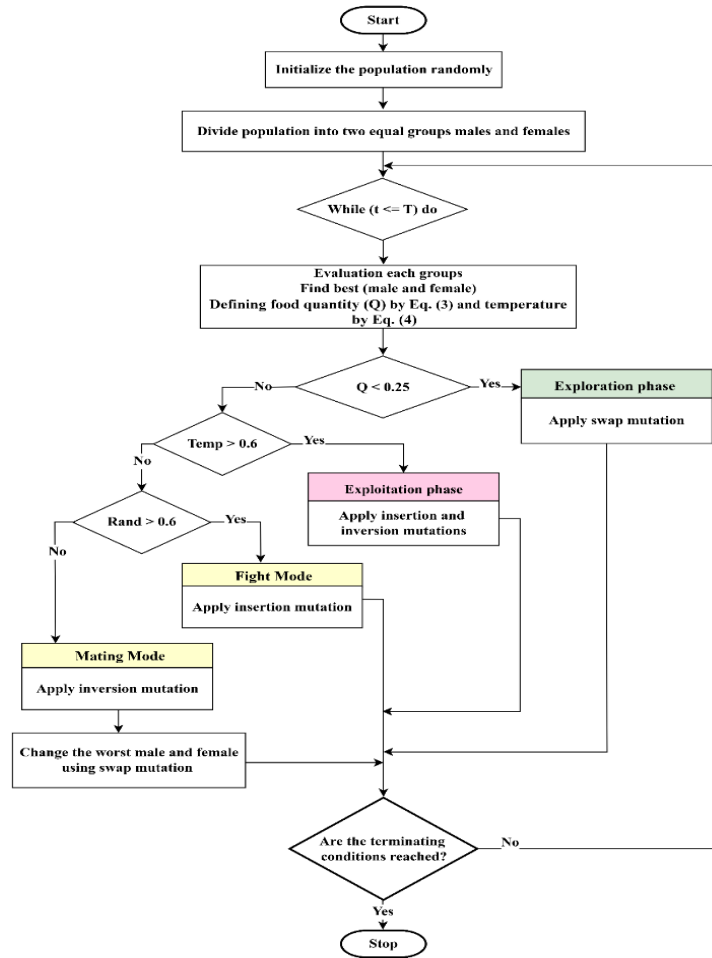


Fig. 2. illustrates the flowchart of the MSO algorithm

6. EXPERIMENTAL RESULTS

The performance of the proposed MSO is assessed in this section. The experimental environment settings are given in Subsection 6.1, and a comparative analysis of MSO, the best-known solution (BKS) and other optimisation algorithms is presented in Subsection 6.2.

6.1. Environment setting

Numerous benchmarks can be adopted for PFSP. However, most researchers employ Taillard's criteria (Taillard, 1993) to evaluate their methods. The benchmarks include 120 cases categorised into 12 scales, each containing 10 instances ranging from 20 jobs and 5 machines to 500 jobs and 20 machines, characterised as $[j * m]$, where $j \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, as shown in Table 1. Taillard benchmark instances are widely used to evaluate algorithms for PFSP. They are categorised into three primary groups based on problem size. Specifically, 20 jobs are included in small instances, and 50 jobs are contained in medium instances. Meanwhile, large instances have 100, 200 and 500 jobs. Each group is tested on 5, 10 and 20 machines, except for the 200-job set, which is available only for 10 and 20 machines, and the 500-job set, which is available only for 20 machines (Table 1). This designed group provides a consistent approach to evaluating algorithm performance across a range of complexity and computational problems. The proposed

MSO was implemented in MATLAB R2021a and run on a machine equipped with an Intel Core i7-8665U CPU at 4.80 GHz. The parameters for all methods were set based on Cai & Chen (2025) and Zhao et al. (2024) and applied consistently across all test cases to enable a reasonable comparison of their performance.

Tab. 1. Taillard benchmarks for (PFSP)

Job Size	5 Machines	10 Machines	20 Machines
20 Jobs	ta01- ta10	ta11- ta20	ta21- ta30
50 Jobs	ta31- ta40	ta41- ta50	ta51- ta60
100 Jobs	ta61- ta70	ta71- ta80	ta81- ta90
200 Jobs	-	ta91- ta100	ta101- ta110
500 Jobs	-	-	ta111- ta120

6.2. Comparison of MSO with BKS and other optimisation algorithms

In this section, the effectiveness of the MSO algorithm is evaluated and compared with other optimisation algorithms in the literature, such as SA, GA, PSO, ISSA (Cai & Chen, 2025), and AdnCSA (Zhao et al., 2024). The evaluation is conducted on Taillard's benchmark instances, which are highly challenging yet widely adopted for PFSP. Table 2 presents several important results. For small problem sizes, the MSO algorithm performs similarly to the other algorithms, consistently achieving the same execution time as the BKS algorithm. This demonstrates the MSO algorithm's flexibility in handling simple scheduling problems without sacrificing quality. For example, in cases Ta001 to Ta006 and Ta011 to Ta027, the results of MSO and BKS are identical, demonstrating the algorithm's ability to converge towards globally optimal solutions in small instances. As the problem size increases, the differences between algorithms become increasingly pronounced. MSO frequently achieves either the best performance or near-best performance compared with its competitors. For instance, MSO obtains lower makespan values than GA, PSO, and AdnCSA in Ta041, Ta042, and Ta045 while still performing on par with ISSA. In large-scale cases such as Ta081–Ta090 and Ta101–Ta120, MSO consistently outperforms SA, GA, and PSO, which perform worse as the search space grows, and produce results on par with BKS. When it comes to problem sizes, MSO stands out for its consistency. In contrast to PSO in Ta051 and Ta074 and GA in Ta083 and Ta101, which sometimes deviate from BKS, MSO consistently achieves competitive results across the benchmark set. For large-scale scheduling, consistency is key, since it is usually more important to maintain solution quality across instances than to excel in a few cases. Other strong algorithms, such as ISSA and AdnCSA, are included in this study. They are surpassed by MSO in several cases. By matching or outperforming these complex algorithms in Ta053-Ta060, MSO demonstrates that it can strike a good balance between exploration and exploitation. Despite challenging circumstances (500×20, Ta111-Ta120), MSO remains competitive and typically achieves the shortest makespan, demonstrating its efficiency and scalability.

Overall, the research demonstrates that MSO provides accurate and reliable solutions to large, complex instances and is just as excellent as BKS at addressing small collections of problems. Because it can outperform both more sophisticated techniques, such as ISSA and AdnCSA, and more conventional algorithms, such as GA, SA, and PSO, MSO is a powerful and adaptable optimisation technique for solving PFSP. Compared to other optimisation algorithms, the proposed MSO algorithm is more effective and powerful. It functions well because it has a flexible exploration mechanism that prevents it from converging too quickly and a powerful exploitation capacity that ensures it converges to nearly optimum solutions. MSO has the potential to be a state-of-the-art PFSP technique, as it consistently yields improved outcomes, particularly in large-scale cases. It is evident from the comparative findings on Taillard's test cases that the intended MSO functions well. MSO consistently receives BKS and performs on par with SA, GA, PSO, ISSA, and AdnCSA for small and medium-sized jobs (such as 20x5, 20x10, and 20x20). This highlights its robustness in solving simple cases. However, as the problem size increases (50x10, 50x20, 100x20, 200x20, and 500x20), the methods differ markedly. In these large-scale situations, the old algorithms (like GA, SA, and PSO) often yield high makespan values, with differences of 1% to 4% relative to BKS. MSO, on the other hand, keeps results within 0.1% to 0.5% of BKS, making the performance gap much smaller. Compared with ISSA and AdnCSA, which are robust contemporary metaheuristics, MSO demonstrates equivalent performance in several instances and even exceeds them on large-scale instances, namely Ta081–Ta090 and Ta111–Ta120.

Tab. 2. Comparison of MSO with BKS and other optimisation algorithms

Problem instance	Size j*m	BKS	MSO	SA	GA	PSO	ISSA	AdnCSA	Deviation (%)
Ta001	20*5	1278	1278	1278	1278	1278	1278	1278	0.00
Ta002		1359	1359	1359	1359	1359	1359	1359	0.00
Ta003		1081	1081	1081	1081	1081	1081	1081	0.00
Ta004		1293	1293	1293	1293	1293	1293	1293	0.00
Ta005		1235	1235	1235	1235	1235	1235	1235	0.00
Ta006		1195	1195	1195	1195	1195	1195	1195	0.00
Ta007		1234	1239	1239	1239	1239	1234	1239	0.00
Ta008		1206	1206	1206	1206	1206	1206	1206	0.00
Ta009		1230	1230	1230	1230	1230	1230	1230	0.00
Ta010		1108	1108	1108	1108	1108	1108	1108	0.00
Ta011	20*10	1582	1582	1582	1582	1582	1582	1582	0.00
Ta012		1659	1659	1659	1659	1659	1659	1659	0.00
Ta013		1496	1496	1496	1496	1496	1496	1496	0.00
Ta014		1377	1377	1377	1377	1377	1377	1377	0.00
Ta015		1419	1419	1419	1419	1419	1419	1419	0.00
Ta016		1397	1397	1397	1397	1397	1397	1397	0.00
Ta017		1484	1484	1484	1484	1484	1484	1484	0.00
Ta018		1538	1538	1538	1538	1543	1538	1538	0.00
Ta019		1593	1593	1593	1593	1593	1593	1593	0.00
Ta020		1591	1591	1591	1591	1598	1591	1591	0.00
Ta021	20*20	2297	2297	2297	2297	2297	2297	2297	0.00
Ta022		2099	2099	2099	2099	2100	2099	2099	0.00
Ta023		2326	2326	2326	2326	2326	2326	2326	0.00
Ta024		2223	2223	2223	2223	2223	2223	2223	0.00
Ta025		2291	2291	2291	2291	2294	2291	2291	0.00
Ta026		2226	2226	2226	2226	2228	2226	2226	0.00
Ta027		2273	2273	2273	2273	2273	2273	2273	0.00
Ta028		2200	2200	2200	2200	2200	2200	2200	0.00
Ta029		2237	2237	2237	2237	2237	2237	2237	0.00
Ta030		2178	2178	2178	2178	2178	2178	2188	0.00
Ta031	50*5	2724	2724	2724	2724	2724	2724	2726	0.00
Ta032		2834	2834	2836	2834	2836	2834	2836	0.00
Ta033		2621	2621	2621	2621	2621	2621	2624	0.00
Ta034		2751	2751	2751	2751	2751	2751	2751	0.00
Ta035		2863	2863	2863	2863	2863	2863	2865	0.00
Ta036		2829	2829	2829	2829	2829	2829	2829	0.00
Ta037		2725	2725	2725	2725	2725	2725	2727	0.00
Ta038		2683	2683	2683	2683	2683	2683	2686	0.00
Ta039		2552	2552	2552	2552	2555	2552	2552	0.00
Ta040		2782	2782	2781	2781	2782	2782	2782	0.00
Ta041	50*10	2991	2994	3024	3021	3051	3023	3021	0.10
Ta042		2867	2870	2882	2902	2915	2873	2902	0.10
Ta043		2839	2839	2852	2871	2889	2852	2859	0.00
Ta044		3063	3063	3063	3070	3071	3063	3071	0.00
Ta045		2976	2988	2982	2998	3024	2984	2998	0.40
Ta046		3006	3006	3006	3024	3036	3006	3018	0.00
Ta047		3093	3100	3122	3122	3133	3107	3122	0.23
Ta048		3037	3040	3042	3063	3049	3039	3052	0.10
Ta049		2897	2911	2911	2914	2923	2902	2923	0.48
Ta050		3065	3069	3077	3076	3131	3078	3098	0.13

Tab. 2. Comparison of MSO with BKS and other optimisation algorithms, continued

Problem instance	Size j*m	BKS	MSO	SA	GA	PSO	ISSA	AdnCSA	Deviation (%)
Ta051	50*20	3771	3850	3889	3874	3950	3883	3893	2.10
Ta052		3668	3714	3714	3734	3761	3715	3716	1.25
Ta053		3591	3648	3667	3688	3741	3672	3676	1.59
Ta054		3635	3755	3754	3759	3806	3748	3755	3.30
Ta055		3553	3610	3644	3644	3688	3636	3644	1.60
Ta056		3667	3681	3708	3717	3758	3693	3695	0.38
Ta057		3672	3711	3754	3728	3763	3722	3728	1.06
Ta058		3627	3700	3711	3730	3788	3721	3730	2.01
Ta059		3645	3780	3772	3779	3831	3761	3762	3.70
Ta060		3696	3772	3778	3801	3830	3777	3767	2.06
Ta061	100*5	5493	5493	5493	5493	5493	5493	5493	0.00
Ta062		5268	5268	5268	5268	5268	5268	5268	0.00
Ta063		5175	5175	5175	5175	5175	5175	5179	0.00
Ta064		5014	5014	5014	5014	5014	5014	5023	0.00
Ta065		5250	5250	5250	5250	5250	5250	5255	0.00
Ta066		5135	5135	5135	5135	5135	5135	5135	0.00
Ta067		5246	5246	5246	5246	5246	5246	5251	0.00
Ta068		5094	5094	5094	5094	5094	5094	5094	0.00
Ta069		5448	5448	5448	5448	5448	5448	5454	0.00
Ta070		5322	5322	5322	5322	5322	5322	5332	0.00
Ta071	100*10	5770	5770	5776	5770	5790	5770	5790	0.00
Ta072		5349	5353	5360	5358	5377	5349	5368	0.07
Ta073		5676	5689	5677	5676	5679	5679	5691	0.23
Ta074		5781	5788	5792	5792	5849	5792	5832	0.12
Ta075		5467	5470	5467	5467	5514	5467	5491	0.05
Ta076		5303	5303	5311	5311	5308	5303	5308	0.00
Ta077		5595	5595	5596	5605	5602	5599	5610	0.00
Ta078		5617	5650	5625	5617	5664	5626	5630	0.59
Ta079		5871	5880	5891	5877	5907	5875	5907	0.15
Ta080		5845	5845	5845	5845	5858	5848	5903	0.00
Ta081	100*20	6106	6260	6257	6303	6414	6278	6272	2.52
Ta082		6183	6203	6226	6266	6383	6245	6278	0.32
Ta083		6252	6310	6342	6351	6437	6330	6333	0.93
Ta084		6254	6301	6303	6360	6407	6303	6304	0.75
Ta085		6262	6387	6380	6408	6509	6378	6386	2.00
Ta086		6302	6411	6427	6453	6551	6458	6451	1.73
Ta087		6184	6304	6306	6332	6476	6314	6316	1.94
Ta088		6315	6468	6472	6482	6640	6464	6546	2.42
Ta089		6204	6302	6380	6343	6462	6327	6396	1.58
Ta090		6404	6488	6485	6506	6593	6488	6570	1.31
Ta091	200*10	10862	10862	10872	10885	10872	10885	10872	0.00
Ta092		10480	10502	10487	10495	10556	10523	10512	0.21
Ta093		10922	10960	10941	10941	10950	10963	10941	0.35
Ta094		10889	10904	10889	10889	10893	10939	10893	0.14
Ta095		10524	10524	10524	10524	10537	10537	10537	0.00
Ta096		10329	10331	10346	10346	10378	10337	10346	0.02
Ta097		10854	10910	10868	10866	10882	10882	10868	0.52
Ta098		10730	10765	10741	10741	10777	10753	10777	0.33
Ta099		10438	10454	10451	10451	10450	10438	10455	0.15
Ta100		10675	10702	10680	10684	10727	10676	10727	0.25
Ta101	200*20	11152	11292	11287	11339	11535	11373	11535	1.26
Ta102		11143	11199	11277	11344	11596	11402	11517	0.50
Ta103		11281	11424	11418	11445	11676	11537	11443	1.27
Ta104		11275	11356	11376	11434	11665	11445	11434	0.72
Ta105		11259	11376	11365	11369	11548	11385	11374	1.04
Ta106		11176	11328	11330	11292	11546	11356	11546	1.36
Ta107		11337	11401	11398	11481	11702	11517	11481	0.56
Ta108		11301	11412	11433	11442	11675	11500	11448	0.98
Ta109		11145	11309	11356	11313	11554	11391	11325	1.47
Ta110		11284	11452	11446	11424	11683	11458	11424	1.49

Tab. 2. Comparison of MSO with BKS and other optimisation algorithms, continued

Problem instance	Size j*m	BKS	MSO	SA	GA	PSO	ISSA	AdnCSA	Deviation (%)
Ta111	500*20	26040	26704	26187	26228	26656	26379	26763	2.55
Ta112		26500	26588	26799	26688	27153	26962	27214	0.33
Ta113		26371	26466	26496	26522	26923	26656	26978	0.36
Ta114		26456	26644	26612	26586	26894	26738	27022	0.71
Ta115		26334	26514	26514	26541	26768	26542	26928	0.68
Ta116		26469	26567	26661	26582	26965	26711	27127	0.37
Ta117		26389	26479	26529	26660	26799	26596	26931	0.34
Ta118		26560	26605	26750	26711	27066	26867	27173	0.17
Ta119		26005	26223	26223	26148	26488	26298	26535	0.84
Ta120		26457	26613	26619	26611	26923	26721	26924	0.59

This finding shows that MSO can achieve an extraordinary equilibrium between exploration and exploitation. This allows MSO to avoid early convergence, which is a problem for GA and PSO, while simultaneously achieving improved stability and solution quality compared to the ISSA and AdnCSA alternatives. The deviation is defined as $= \frac{MSO-BKS}{BKS} \times 100\%$. The MSO algorithm in Table 2 achieved the best results in 57 of 120 problem instances. Overall, MSO is a more powerful and reliable optimiser that offers higher accuracy, greater scalability, and more consistent performance across diverse PFSP instances than its state-of-the-art competitors. The results of the nonparametric Friedman test and Holm's post hoc analysis, both of which were used for statistical research, are shown in Table 3.

Tab. 3. Results based on the average of all the performances

Minimizing makespan	Algorithms					
	MSO	SA	GA	PSO	ISSA	AdnCSA
	2.65	2.87	3.28	4.73	3.18	4.26

In the context of reducing the average makespan, the purpose of this statistical test is to determine whether or not the suggested approach performs significantly better than other optimisation methods. As shown in Table 3, the average outcomes for all PFSP instances are shown. Additionally, this table includes the average shortest possible makespan. Among all the optimisation strategies tested, the presented MSO has the shortest makespan.

7. CONCLUSIONS

To address the PFSP and minimise makespan, an MOS was proposed in this study. The algorithm integrates three mutation strategies (swap, insertion, and inversion) that enhance exploration and exploitation by diversifying the search and preventing premature convergence. Experimental evaluations on Taillard's benchmark sets confirmed MSO's strong performance across small, medium, and large instances. For smaller problems, MSO consistently achieved the best-known solutions, matching those of state-of-the-art algorithms such as SA, GA, PSO, ISSA, and AdnCSA. More importantly, for large-scale instances, where traditional algorithms often deviated by 1–4% from the BKS, MSO maintained results within 0.1–0.5%, often outperforming optimisation methods. These results indicate the robustness, scalability, and efficiency of the MSO algorithm as a reliable method for solving the PFSP. To further enhance solution quality, future research could extend MSO to incorporate hybrid or multi-objective scheduling approaches.

Conflicts of Interest

The authors declare no conflict of interest.

REFERENCES

- Al-Amin, M., Baldacci, R., & Kayvanfar, V. (2025). A comprehensive review on operating room scheduling and optimization. *Operational Research*, 25(1), 1–30. <https://doi.org/10.1007/s12351-024-00850-x>

- Al-Attabi, K. (2022). Snake Optimization with deep learning enabled disease detection model for colorectal cancer. *Journal of Smart Internet of Things*, 2022(1), 178–195. <https://doi.org/10.2478/jsiot-2022-0012>
- Al-Shourbaji, I., Kachare, P. H., Alshathri, S., Duraibi, S., Elnaim, B., & Elaziz, M. A. (2022). An efficient parallel reptile search algorithm and snake optimizer approach for feature selection. *Mathematics*, 10(13), Article 2351. <https://doi.org/10.3390/math10132351>
- Almazini, H. F., Mortada, S., Al-Mazini, H. F. A., Al-Behadili, H. N. K., & Alkenani, J. (2022). Improved discrete plant propagation algorithm for solving the traveling salesman problem. *IAES International Journal of Artificial Intelligence*, 11(1), 13–22. <https://doi.org/10.11591/ijai.v11.i1.pp13-22>
- Arbaoui, B., Wahid, J., & Abdul-Rahman, S. (2022). A modified great deluge algorithm with dual decay rate for school timetabling. *International Journal of Intelligent Engineering and Systems*, 15(6), 479–491. <https://doi.org/10.22266/ijies2022.1231.43>
- Blum, C., & Sampels, M. (2002). Ant colony optimization for FOP shop scheduling: A case study on different pheromone representations. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)* (Vol. 2, pp. 1558–1563). IEEE. <https://doi.org/10.1109/CEC.2002.1004475>
- Cai, Y., & Chen, H. (2025). An improved salp swarm algorithm for permutation flow shop vehicle routing problem. *Scientific Reports*, 15(1), Article 86054. <https://doi.org/10.1038/s41598-025-86054-3>
- Fong, S., Lou, H.-L., Zhuang, Y., Deb, S., & Hanne, T. (2014). Solving the permutation flow shop problem with firefly algorithm. In *2014 2nd International Symposium on Computational and Business Intelligence* (pp. 25–29). IEEE. <https://doi.org/10.1109/ISCBI.2014.13>
- Gogos, C. (2023). Solving the distributed permutation flow-shop scheduling problem using constrained programming. *Applied Sciences*, 13(23), Article 12562. <https://doi.org/10.3390/app132312562>
- Hashim, F. A., & Hussien, A. G. (2022). Snake Optimizer: A novel meta-heuristic optimization algorithm. *Knowledge-Based Systems*, 242, Article 108320. <https://doi.org/10.1016/j.knsys.2022.108320>
- Huang, K. W., Wu, Z. X., Jiang, C. L., Huang, Z. H., & Lee, S. H. (2023). WPO: A whale particle optimization algorithm. *International Journal of Computational Intelligence Systems*, 16(1), Article 45. <https://doi.org/10.1007/s44196-023-00295-6>
- Jabbar, A. M., & Ku-Mahamud, K. R. (2021). Grey wolf optimization algorithm for hierarchical document clustering. *Indonesian Journal of Electrical Engineering and Computer Science*, 24(3), 1744–1758. <https://doi.org/10.11591/ijeecs.v24.i3.pp1744-1758>
- Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1), 61–68. <https://doi.org/10.1002/nav.3800010110>
- Khalilpourazari, S., & Khalilpourazary, S. (2018). Optimization of production time in the multi-pass milling process via a Robust Grey Wolf Optimizer. *Neural Computing and Applications*, 29(12), 1321–1336. <https://doi.org/10.1007/s00521-016-2644-6>
- Liang, Z., Zhong, P., Liu, M., Zhang, C., & Zhang, Z. (2022). A computational efficient optimization of flow shop scheduling problems. *Scientific Reports*, 12(1), Article 1248. <https://doi.org/10.1038/s41598-022-04887-8>
- Liu, Y. F., & Liu, S. Y. (2013). A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing*, 13(3), 1459–1463. <https://doi.org/10.1016/j.asoc.2011.10.024>
- Lu, H., Zhan, H., & Wang, T. (2024). A multi-strategy improved snake optimizer and its application to SVM parameter selection. *Mathematical Biosciences and Engineering*, 21(10), 7297–7336. <https://doi.org/10.3934/mbe.2024322>
- Mortada, S. (2026). Binary border collie optimization algorithm for feature selection. *International Arab Journal of Information Technology*, 23(1), 48–57. <https://doi.org/10.34028/iajit/23/1/5>
- Mzili, T., Mzili, I., Riffi, M. E., Pamucar, D., Simic, V., & Kurdi, M. (2023). A novel discrete rat swarm optimization algorithm for the quadratic assignment problem. *Facta Universitatis, Series: Mechanical Engineering*, 21(3), 529–552. <https://doi.org/10.22190/FUME230602024M>
- Ni, P., Su, X., Fu, J., & Bai, Y. (2025). A hybrid snake optimizer with crisscross learning strategy for constrained structural optimization. *Engineering Optimization*, 57(3), 512–534. <https://doi.org/10.1080/0305215X.2025.2469664>
- Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5), 461–476. <https://doi.org/10.1016/j.omega.2004.12.006>
- Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2), 278–285. [https://doi.org/10.1016/0377-2217\(93\)90182-M](https://doi.org/10.1016/0377-2217(93)90182-M)
- Tasgetiren, M. F., Sevkli, M., Liang, Y. C., & Gencyilmaz, G. (2004). Particle swarm optimization algorithm for permutation flowshop sequencing problem. *European Journal of Operational Research*, 177(3), 1930–1947. <https://doi.org/10.1016/j.ejor.2005.12.028>
- Zhao, C., Wu, L., Zuo, C., Zhang, H., & Xiao, Q. (2024). An adaptive dynamic neighborhood crow search algorithm for solving permutation flow shop scheduling problems. *Journal of Industrial and Management Optimization*, 20(1), 84–111. <https://doi.org/10.3934/jimo.2023052>
- Zheng, K., Liu, H., & Li, B. (2025). Improved Snake Optimization Algorithm for global optimization and engineering applications. *Scientific Reports*, 15(1), Article 1299. <https://doi.org/10.1038/s41598-025-01299-2>