# The Power of a Collective: Team of Agents Solving Instances of the Flow Shop and Job Shop Problems

Piotr Jedrzejowicz[0000−0001−6104−1381] and Izabela Wierzbowska[0000−0003−4818−4841]

Gdynia Maritime University, Gdynia, Poland
p.jedrzejowicz@umg.edu.pl, i.wierzbowska@wpit.umg.edu.pl

**Abstract.** The paper proposes an approach for solving difficult combinatorial optimization problems integrating the mushroom picking population-based metaheuristic, a collective of asynchronous agents, and a parallel processing environment, in the form of the MPF framework designed for the Apache Spark computing environment. To evaluate the MPF performance we solve instances of two well-known NP-hard problems – job shop scheduling and flow shop scheduling. In MPF a collective of simple agents works in parallel communicating indirectly through the access to the common memory. Each agent receives a solution from this memory and writes it back after a successful improvement. Computational experiment results confirm that the proposed MPF framework can offer competitive results as compared with other recently published approaches.

**Keywords:** Collective of Agents · Metaheuristics · Parallel Computations · Computationally Hard Combinatorial Optimization Problems

## 1 Introduction

Computational collective intelligence (CCI) techniques use computer-based models, algorithms, and tools that take advantage of the synergetic effects of interactions between agents acting in parallel to reach a common goal. In the field of optimization, applications of the CCI techniques usually involve the integration of multiple agent systems with the population-based metaheuristics including the cooperative co-evolutionary algorithms.

Population-based metaheuristics are used to deal with computationally difficult optimization problems like, for example, combinatorial optimization, global optimization in complex systems, multi-criteria optimization as well as optimization and control in dynamic systems. Population in a population-based metaheuristic represents solutions or some constructs that can be easily transformed into solutions. Population-based algorithms reach their final solutions after having carried out various operations transforming populations, sub-populations, or population members to find the best solution. Advantages of the population-based algorithms can be attributed to their following abilities:

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

- Reviewing in a reasonable time a big number of possible solutions from the search space.
- Directing search processes towards more promising areas of the search space.
- Increasing computation effectiveness through implicit or explicit cooperation between population members and thus achieving a synergetic effect.
- Performing a search for the optimum solution in parallel and a distributed environment.

More details on the population-based metaheuristics can be found in reviews of [6], [12] and [20].

An important tool for increasing computation effectiveness in solving difficult optimization problems is the decentralization of efforts and cooperation between decentralized computational units. To achieve full advantages of such a cooperation, multiple agent frameworks have been proposed and implemented. Agent-based implementation of metaheuristics allows autonomous agents to communicate and cooperate through information exchange synchronously or asynchronously. Besides, there might be some kind of learning implemented in agents. This feature enables the agent to assimilate knowledge about the environment and other agents' actions and use it to improve the consequences of their actions. The review of frameworks for the hybrid metaheuristics and multi-agent systems for solving optimization problems can be found in [19]. Example frameworks used for developing multi-agent systems and implementing population-based metaheuristic algorithms include AMAM - a multi-agent framework applied for solving routing and scheduling problems [18] and JABAT, a middleware for implementing JADE-based and population-based A-Teams [3].

Effects of integrating population-based metaheuristics and multi-agent technology for solving difficult computational problems, especially combinatorial optimization problems, are constrained by the available computation technologies. Recent developments in the field of parallel and distributed computing make it possible to alleviate some of these constraints. Several years ago parallel and distributed computing were a promising, but rather a complex way of programming. At present every programmer should have a working knowledge of these paradigms, to exploit current computing architectures [8].

This paper aims to show that integrating an approach involving a population-based metaheuristic, a collective of asynchronous agents, and a parallel processing environment, may benefit the search for a solution in case of difficult combinatorial optimization problems. To demonstrate that the above statement holds we show the results of a computational experiment involving parallel implementation of the Mushroom Picking Algorithm (MPA) with asynchronous agents. Our test-bed consists of two well-known NP-hard scheduling problems – flow shop (PFSP) and job shop (JSSP), and the MPF framework designed to enable MPA implementation using the Apache Spark, an open-source data-processing engine for large data sets. It is designed to deliver the computational speed, scalability, and programmability required for Big Data [22].

The rest of the paper is constructed as follows. Section 2 contains a brief description of the considered scheduling problems. Section 3 reviews currently

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

published algorithms for solving instances of PFSP and JSSP. Section 4 gives details of the implementation of the parallel, agent-based, using the MPF framework. Section 5 contains the results of the computational experiment. Section 6 includes conclusions and suggestions for future research.

## 2 Scheduling Problems

**Job Shop Scheduling Problem (JSSP)** consists of a set of $n$ jobs $(j_1, \ldots, j_n)$ to be scheduled on $m$ machines $(m_1, \ldots, m_m)$. Each job consists of operations (tasks) that have to be processed in the given order. Each operation within a job must be processed on a specific machine, only after all preceding operations of this job are completed.

Further constraints include:

- Operations cannot be interrupted.
- Each machine can handle only one job at a time.

The goal is to find the job sequences on machines minimizing the makespan. A single solution may be represented as the ordered list of the numbers of the jobs. The length of the list is $n \times m$. There are $m$ occurrences of each job in such a list. When examining the list from the left to the right, the $i$th occurrence of job $j$ refers to the $i$th operation (task) of this job.

The problem was proven to be NP-hard in [16].

**Permutation Flow Shop Scheduling Problem (PFSP)** consists of a set of different machines that carry out operations (tasks) of jobs. All jobs have identical processing order of their operations. Following [5], assume that the order of processing a set of jobs $J$ on $m$ different machines is described by the machine sequence $P_1, \ldots, P_m$. Hence, job $J_j \in J$ consists of $m$ operations $O_{1j}, \ldots, O_{mj}$ with processing times $p_{ij}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$ where $n$ is the number of jobs in $J$.

The following constraints on jobs and machines have to be met:

- Operations cannot be interrupted.
- Each machine can handle only one job at a time.

While the machine sequence of all jobs is identical, the problem is to find the job sequence minimizing the makespan (maximum of the completion times of all tasks). A single solution may be represented as the ordered list of the numbers of the jobs of the length $n$.

The problem was proven to be NP-hard in [10].

## 3 Current approaches for solving PFSP and JSSP

### 3.1 Algorithms and approaches for solving JSSP instances

Population-based algorithms including swarm intelligence and evolutionary systems have proven successful in tackling JSSP, one of the hard optimization problems considered in this study. A state of the art review on the application of the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

AI techniques for solving the JSSP as of 2013 can be found in [29]. Recently, several interesting swarm intelligence solutions for JSSP were published. In [11] the local search mechanism of the PSA and large-span search principle of the cuckoo search algorithm are combined into an improved cuckoo search algorithm (ICSA). A hybrid algorithm for solving JSSP integrating PSO and neural network was proposed in [36]. An improved whale optimization algorithm (IWOA) based on quantum computing for solving JSSP instances was proposed by [37]. An improved GA for JSSP [7] offers good performance. Their niche adaptive genetic algorithm (NAGA) involves several rules to increase population diversity and adjust the crossover rate and mutation rate according to the performance of the genetic operators. Niche is seen as the environment permitting species with similar features to compete for survival in the elimination process. According to the authors, the niche technique prevents premature convergence and improves population diversity. Recently, a well-performing GA for solving JSSP instances was proposed in [14]. The authors suggest a feasibility preserving solution representation, initialization, and operators for solving job-shop scheduling problems. Another genetic algorithm combined with the local search was proposed in [30]. The approach features the use of a local search strategy in the traditional mutation operator; and a new multi-crossover operator.

A novel two-level metaheuristic algorithm was suggested in [21]. The lower-level algorithm is a local search algorithm searching for an optimal JSSP solution within a hybrid neighborhood structure. The upper-level algorithm is a population-based search algorithm developed for controlling the input parameters of the lower-level algorithm.

A discrete wolf pack algorithm (DWPA) for job shop scheduling problems was proposed in [32]. DWPA involves 3 phases: initialization, scouting, and summoning. During initialization heuristic rules are used to generate a good quality initial population. The scouting phase is devoted to the exploration while summoning takes care of the intensification. In [31] a novel biomimicry hybrid bacterial foraging optimization algorithm (HBFOA) was developed. HBFOA is inspired by the behavior of E. coli bacteria in its search for food. The algorithm is hybridized with simulated annealing. Additionally, the algorithm was enhanced by a local search method. Evaluation of the performance of several PSO-based algorithms for solving the JSSP can be found in [1].

As in the case of other computationally difficult optimization problems, an emerging technology supported development of parallel and distributed algorithms for solving JSSP instances. A scheduling algorithm, called MapReduce coral reef (MRCR) for JSSP instances was proposed in [28]. The basic idea of the proposed algorithm is to apply the MapReduce platform and the Spark Apache environment to implement the coral reef optimization algorithm to speed up its response time. More recently, a large-scale flexible JSSP optimization by a distributed evolutionary algorithm was proposed in [25]. The algorithm belongs to the distributed cooperative evolutionary algorithms class and is implemented on Apache Spark.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

### 3.2 Algorithms and approaches for solving PFSP instances

There exist several heuristic approaches for solving PFSP. Selected heuristics, namely CDS, Palmer's slope index, Gupta's algorithm, and concurrent heuristic algorithm for minimizing the makespan in permutation flow shop scheduling problem were studied in [24]. An improved heuristic algorithm for solving the flow shop scheduling problem was proposed in [23]. In [4] the adapted Nawaz-Enscore-Ham (NEH) heuristic and two metaheuristics based on the exploration of the neighborhood are studied. Another modification of NEH heuristic was suggested in [17] where a novel tie-breaking rule was developed by minimizing partial system idle time without increasing the computational complexity of the NEH heuristic.

A Tabu Search with the intensive concentric exploration over non-explored areas was proposed in [9] as an alternative solution to the simplest Tabu Search with the random shifting of two jobs indexes operation for Permutation Flow Shop Problem (PFSP) with the makespan minimization criterion.

Recently, several metaheuristics have proven effective in solving PFSP instances. In [33] the authors propose two water wave optimization (WWO) algorithms for PFSP. The first algorithm adapts the original evolutionary operators of the basic WWO. The second further improves the first algorithm with a self-adaptive local search procedure. Application of the cuckoo search metaheuristic for PFSP was suggested in [35]. The approach shows good performance in solving the permutation flow shop scheduling problem. Modified Teaching-Learning-Based Optimization with Opposite-Based-Learning algorithm was applied to solve the Permutation Flow-Shop-Scheduling Problem under the criterion of minimizing the makespan was proposed in [2]. To deal with the complex PFSPs, the paper of [34] proposed an improved simulated annealing (SA) algorithm based on the residual network. First, this paper defines the neighborhood of the PFSP and divides its key blocks. Second, the residual network algorithm is used to extract and train the features of key blocks. Next, the trained parameters are used in the SA algorithm to improve its performance.

## 4 An approach for solving JSSP and PFSP

### 4.1 The MPF framework

To deal with the considered combinatorial optimization problems we use the Mushroom Picking Framework (MPF). The MPF is based on the Mushroom Picking Algorithm (MPA) originally proposed in [13] for solving instances of the Traveling Salesman Problem and job shop scheduling. The metaphor of MPA refers to a situation where many mushroom pickers, with different preferences as to the collected mushroom kinds, explore the woods in parallel pursuing individual or random, or mixed strategies and trying to increase the current crop. Pickers exchange information indirectly by observing traces left by others and modifying their strategies accordingly. In case of finding interesting species, they intensify search in the vicinity hoping to find more specimens. In the MPA a set

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

of simple, dedicated, agents, metaphorically mushroom pickers, explore in parallel the search space. Agents differ between themselves by performing different operations on the encountered solutions. They may have also different computational complexities. Agents explore a search space randomly intensifying their efforts after having found an improved solution.

MPF differs from MPA in being only a framework, allowing the user to define the internal algorithm controlling a solution improvement processes performed by an agent. There are no constraints on the number of agents with different internal algorithms used. There are also no constraints on the overall number of agents employed for solving a particular instance of the problem at hand. The user is also responsible for generating the initial population of solutions and for storing it in the common memory. MPF provides the capability of reading one or more solutions from the common memory and the capability of writing an improved by an agent solution in the common memory.

Agents in the MPF work in parallel, in threads, and cycles. Each cycle involves the following steps:

- Solutions in the common memory are randomly shuffled.
- The population of solutions in the common memory is divided into several subpopulations of roughly equal size. Observe that shuffling at stage I assures that subpopulations do not consist of the same solutions in different cycles.
- Each subpopulation is processed by a set of agents in a separate thread. The same composition of agent kinds and numbers is used in each thread. Each agent receives a solution or solutions (depending on the number of arguments of the agent) and runs its internal algorithm which could be, for example, a local search algorithm, to produce an improved solution. If such a solution is found, it replaces the solution drawn from the subpopulation in the case of the single argument agents. Otherwise, it replaces the worst one, out of all solutions processed by an agent.
- The cycle ends after a predefined number of trials to improve the subpopulations have been applied in all threads.
- At the end of a cycle all current subpopulations are appended into the common memory.

The overall stopping criterion is defined as no improvement of the best result (fitness) after the predefined number of cycles has elapsed.

### 4.2 The MPF framework implementation for scheduling problems

The general scheme of the MPF implementation for scheduling problems is shown in a pseudo-code as Algorithm 1.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

**Algorithm 1:** MPA

```
n ← the number of parallel threads
solutions ← a set of solutions with empty sequence of jobs
while !stoppingCriterion do
    populations ← solutions randomly split into n subsets of equal size
    populationsRDD ← populations parallelized in ApacheSpark
    populationsRDD ← populationsRDD.map(p =>
      p.applyOptimizations)
    solutions = populationsRDD.flatMap(identity).collect()
                    // thanks to flatMap, collect returns list
                    // of solutions, not list of populations
    bestSolution ← a solution from solutions with the best fitness
return bestSolution
```

In Algorithm 1, *applyOptimization* is responsible for improving solutions in each subpopulation in all threads. In the first cycle, *ApplyOptimizations* receives solutions not yet initialized as for the sequence of jobs, and it starts with filling these solutions with randomly generated sequences of jobs.

For the proposed implementation of the MPF for solving PFSP and JSSP instances in each thread we use the following set of agents:

- randomReverse — takes a random slice of the list of jobs and reverses the order of its elements;
- randomMove – takes one random job from the list of jobs and moves it to another, random position,
- randomSwap – replaces jobs on two random positions in the list of jobs,
- crossover – requires two solutions. A slice from the first solution is extended with the missing jobs in the order as in the second solution.

During computations, solutions in each subpopulation may, with time, become similar or even the same. To assure the required level of diversification of the solutions, two measures are introduced:

- The crossover agent is chosen by the *ApplyOptimization* procedure twice less often than each of the one-argument agents (in each thread there is only one such agent, while the other agents come in pairs).
- If two solutions drawn for the crossover agent have the same fitness, or fitness differing by 1, the worse solution is replaced by a new random one.

*ApplyOptimization* is shown as Algorithm 2.

The implementation for both considered problems that is PFSP and JSSP differs mainly in how solutions are represented as explained in Subsection 2 and Subsection 2. In both cases, a solution is represented by a list of numbers and such solutions are processed in the same way in all subpopulations, and by the same agents, as described earlier.

If a method of calculating the length of the makespan is defined for the JSSP, then it may be also used for PFSP, however first the solution of PFSP must be

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

**Algorithm 2:** applyOptimizations

```
solutions ← solutions in the subpopulation
foreach s ∈ solutions do
    if s.jobs == null then                // s has empty sequence of jobs
        s.jobs ← random sequence of jobs
for k ← 1 to given number of iterations do
    A ← random agent from the available agents
    if A is two argument agent then
        s1, s2 ← two solutions drawn from solutions
        sw ← s1 max s2        // solution with the bigger makespan
        if abs(s1.makespan–s2.makespan) < 2 then
            in solutions replace sw with a random solution
        else
            newSolution ← A(s1, s2)
            if newSolution.makespan < sw.makespan then
                in solutions replace sw with newSolution
    else
        s ← draw one solution from solutions
        newSolution ← A(s)
        if newSolution.makespan < s.makespan then
            in solutions replace s with newSolution
return solutions
```

transformed to represent the sequence of operations as in the JSSP case. The solution $(j_1, j_2, \ldots, j_n)$ is mapped to $(j_1, j_1, \ldots, j_1, j_2, j_2, \ldots, j_2, \ldots, j_n, j_n, \ldots, j_n)$. Thus the same code with very few changes (including the mapping procedure) has been used for both problems.

## 5 Computational Experiment Results

To validate the proposed approach, we have carried out several computational experiments. Experiments were based on two widely used benchmark datasets: the Lawrence dataset for JSSP [15], and the Taillard dataset for PFSP [27]. Both datasets contain instances with known optimal solutions for the minimum makespan criterion. All computations have been run on Spark cluster consisting of 8 nodes with 32 virtual central processing units at the Academic Computer Center in Gdansk. Performance measures included errors calculated as a percentage deviation from the optimal solution value and computation time in seconds.

In [13] it has been shown, that in the MPA the choice of agents that are used to improve solutions may lead to significant differences in the produced results. For the current MPF implementation, agents have been redesigned and changed as described in Subection 4.2. In Table 1 the performance of the proposed approach denoted as MPF is compared with results from [13] and performances of other recently published algorithms for solving JSSP on Lawrence benchmark instances. The errors for [14] have been calculated based on the results from their paper. The results for MPF have been averaged over 30 runs for each problem

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

instance. For solving the JSSP instances by the MPF, the following parameter settings have been used:

- for instances from la01 to la15 - 200 subpopulations, each consisting of 3 solutions, 3000 iterations in each cycle and stopping criterion as no change in the best solution for two consecutive cycles;
- for instances from la16 to la40 - 400 subpopulations, each consisting of 3 solutions, 6000 iterations in each cycle, and stopping criterion as no change in the best solution for five consecutive cycles.

From Table 1 it can be observed that MPF outperforms in terms of both measures - average error and computation time - MPA, GA of [14], enhanced GA of [30]. The enhanced two-level metaheuristic (MUPLA) of [21] offers smaller average errors at the cost of exceedingly high computation times.

To gain better insight into factors influencing the performance of the proposed approach we have run several variants of MPF with different components using a sample of instances from the Lawrence benchmark dataset as shown in Table 2. These experiments were run with the same parameter settings as in the case of results shown in Table 1.

From the results shown in Table 2, it can be observed that both mechanisms introduced within the proposed approach, that is shuffling of solutions in the common memory, and diversification by introducing random solutions, enhance the performance of the MPF. Shuffling stands behind the indirect cooperation between agents and both – diversification and shuffling help getting out of local optima. It should be also noted that results produced by MPF are fairly stable in terms of the average standard deviation of errors.

The PFSP problem experiment has been based on the Taillard benchmark dataset consisting of 10 instances for each considered problem size. Best known values for Taillard instances can be found online [26]. In the experiment the following settings for the proposed MPF have been used:

- for sizes 200x20 and 500x20 - 112 three-solution subpopulations, 100 iterations in each cycle and stopping criterion as no change in the best solution for 10 and 5 consecutive cycles respectively;
- for 50x20, 100x10, 100x20, 200x10 - 200 three-solution subpopulations, 1500 iterations in each cycle and stopping criterion as no change in the best solution for 5 consecutive cycles;
- for all other instances – 200 three-solution subpopulations, 3000 iterations in each cycle, and stopping criterion as no change in the best solution for 5 consecutive cycles;

In Table 3 the performance of MPF is compared with results of other, recently published, approaches.

From the results shown in Table 3, it can be observed that in terms of average error outperforms other approaches except for the water wave optimization algorithm implementation of [33]. Unfortunately, information as to computation times is not available for other approaches. The average standard deviation of errors in the case of the MPF is fairly stable for smaller instances, growing with the problem size.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

**Table 1.** Comparison of results for the JSSP problem

| | | MPF | | | MPA [13] | | GA[14] | mXLSGA [30] | | MUPLA [21] | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Data-set | Make-span | Error % | Time s | SD % | Error % | Time s | Error % | Error % | Time s | Error % | Time s |
| la01 | 666 | 0.00% | 1 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 1 |
| la02 | 655 | 0.00% | 1 | 0.00% | 0.00% | 1 | 1.22% | 0.00% | n.a. | 0.00% | 2 |
| la03 | 597 | 0.41% | 2 | 0.54% | 0.84% | 2 | 1.01% | 0.00% | 34 | 0.00% | 10 |
| la04 | 590 | 0.00% | 1 | 0.00% | 0.24% | 1 | 2.37% | 0.00% | n.a. | 0.00% | 2 |
| la05 | 593 | 0.00% | 1 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 0 |
| la06 | 926 | 0.00% | 1 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 2 |
| la07 | 890 | 0.00% | 1 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 2 |
| la08 | 863 | 0.00% | 1 | 0.00% | 0.00% | 1 | 3.23% | 0.00% | n.a. | 0.00% | 2 |
| la09 | 951 | 0.00% | 1 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 2 |
| la10 | 958 | 0.00% | 1 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 2 |
| la11 | 1222 | 0.00% | 2 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 4 |
| la12 | 1039 | 0.00% | 2 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 8 |
| la13 | 1150 | 0.00% | 2 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 6 |
| la14 | 1292 | 0.00% | 2 | 0.00% | 0.00% | 1 | 0.00% | 0.00% | n.a. | 0.00% | 6 |
| la15 | 1207 | 0.00% | 3 | 0.00% | 0.00% | 31 | 0.75% | 0.00% | n.a. | 0.00% | 7 |
| la16 | 945 | 0.07% | 21 | 0.05% | 0.31% | 41 | 2.96% | 0.00% | n.a. | 0.00% | 294 |
| la17 | 784 | 0.01% | 16 | 0.07% | 0.06% | 40 | 1.66% | 0.00% | 70 | 0.00% | 33 |
| la18 | 848 | 0.00% | 21 | 0.00% | 0.20% | 41 | 2.48% | 0.00% | n.a. | 0.00% | 24 |
| la19 | 842 | 0.20% | 26 | 0.35% | 1.01% | 42 | 4.87% | 0.00% | n.a. | 0.00% | 149 |
| la20 | 901 | 0.46% | 16 | 0.21% | 0.50% | 33 | 1.77% | 0.00% | n.a. | 0.00% | 1073 |
| la21 | 1046 | 1.55% | 64 | 0.58% | 3.03% | 79 | 10.07% | 1.24% | n.a. | 0.06% | 30668 |
| la22 | 927 | 1.23% | 63 | 0.42% | 2.08% | 75 | 11.00% | 0.86% | n.a. | 0.00% | 1439 |
| la23 | 1032 | 0.00% | 26 | 0.00% | 0.00% | 45 | 5.72% | 0.00% | n.a. | 0.00% | 25 |
| la24 | 935 | 1.68% | 57 | 0.78% | 3.50% | 65 | 10.37% | 1.17% | n.a. | 0.26% | 21350 |
| la25 | 977 | 1.60% | 67 | 0.74% | 3.52% | 78 | 8.50% | 0.92% | n.a. | 0.00% | 15827 |
| la26 | 1218 | 0.13% | 88 | 0.27% | 1.61% | 106 | 11.17% | 0.00% | n.a. | 0.00% | 82 |
| la27 | 1235 | 3.21% | 85 | 0.56% | 4.49% | 121 | 13.52% | 2.75% | n.a. | 0.03% | 194427 |
| la28 | 1216 | 1.90% | 117 | 0.65% | 3.15% | 114 | 13.65% | 1.89% | n.a. | 0.00% | 1972 |
| la29 | 1152 | 5.69% | 114 | 1.05% | 7.77% | 121 | 16.58% | 4.26% | n.a. | 1.02% | 130059 |
| la30 | 1355 | 0.01% | 68 | 0.05% | 0.61% | 112 | 9.30% | 0.00% | 236 | 0.00% | 123 |
| la31 | 1784 | 0.00% | 64 | 0.00% | 0.00% | 64 | 3.25% | 0.00% | n.a. | 0.00% | 306 |
| la32 | 1850 | 0.00% | 76 | 0.00% | 0.00% | 85 | 5.46% | 0.00% | n.a. | 0.00% | 172 |
| la33 | 1719 | 0.00% | 60 | 0.00% | 0.00% | 74 | 4.07% | 0.00% | n.a. | 0.00% | 313 |
| la34 | 1721 | 0.00% | 87 | 0.00% | 0.36% | 172 | 7.50% | 0.00% | n.a. | 0.00% | 448 |
| la35 | 1888 | 0.00% | 71 | 0.00% | 0.03% | 95 | 3.92% | 0.00% | n.a. | 0.00% | 393 |
| la36 | 1268 | 3.23% | 87 | 0.72% | 4.53% | 80 | 10.02% | 2.12% | n.a. | 0.00% | 85418 |
| la37 | 1397 | 3.84% | 87 | 0.96% | 4.94% | 94 | 13.10% | 1.28% | n.a. | 0.00% | 60481 |
| la38 | 1196 | 4.27% | 116 | 1.17% | 7.02% | 106 | 17.56% | 4.18% | n.a. | 0.25% | 169974 |
| la39 | 1233 | 2.39% | 108 | 0.45% | 4.25% | 99 | 12.08% | 2.02% | n.a. | 0.00% | 18057 |
| la40 | 1222 | 2.61% | 102 | 0.99% | 3.69% | 109 | 13.26% | 1.71% | n.a. | 0.16% | 119463 |
| avg | | 0.86% | 43 | 0.00% | 1.44% | 55 | 5.56% | 0.61% | n.a. | 0.04% | 21316 |

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

**Table 2.** MPF performance with different variants of components used

| Dataset | random, shuffling Error % | Time s | SD % | no random, shuffling Error % | Time s | SD % | random, no shuffling Error % | Time s | SD % | no random, no shuffling Error % | Time s | SD % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| la20 | 0.46% | 16 | 0.2% | 0.48% | 14 | 0.2% | 0.46% | 14 | 0.2% | 0.63% | 15 | 0.2% |
| la21 | 1.55% | 64 | 0.6% | 2.37% | 43 | 0.8% | 3.52% | 54 | 0.8% | 4.65% | 39 | 0.7% |
| la22 | 1.23% | 63 | 0.4% | 1.70% | 47 | 0.4% | 2.44% | 64 | 0.6% | 3.32% | 42 | 0.9% |
| la23 | 0.00% | 26 | 0.0% | 0.00% | 26 | 0.0% | 0.00% | 34 | 0.0% | 0.00% | 32 | 0.0% |
| la24 | 1.68% | 57 | 0.8% | 2.74% | 53 | 0.7% | 3.28% | 57 | 0.7% | 4.72% | 46 | 0.8% |
| la25 | 1.60% | 67 | 0.7% | 2.81% | 57 | 1.0% | 3.22% | 59 | 1.0% | 5.36% | 42 | 0.7% |
| la26 | 0.13% | 88 | 0.3% | 0.35% | 89 | 0.5% | 1.42% | 113 | 0.9% | 2.61% | 88 | 0.9% |
| la27 | 3.21% | 85 | 0.6% | 3.58% | 84 | 0.6% | 4.96% | 95 | 0.6% | 5.66% | 78 | 0.8% |
| la28 | 1.90% | 117 | 0.6% | 2.20% | 96 | 0.5% | 3.75% | 106 | 0.6% | 4.90% | 83 | 1.0% |
| la29 | 5.69% | 114 | 1.1% | 6.56% | 106 | 1.3% | 8.45% | 90 | 0.5% | 9.67% | 83 | 0.9% |
| la30 | 0.01% | 68 | 0.1% | 0.11% | 91 | 0.3% | 0.52% | 106 | 0.5% | 1.92% | 77 | 0.8% |
| avg | 1.59% | 70 | 0.5% | 2.08% | 64 | 0.6% | 2.91% | 72 | 0.6% | 3.95% | 57 | 0.7% |

**Table 3.** Performance of the MPF versus other approaches

| Size | MPF Error % | MPF Time s | MPF SD % | CH [24] Error % | [23] Error % | NEHLJP1 [17] Error % | WWO [33] Error % |
|---|---|---|---|---|---|---|---|
| 20x5 | 0.04% | 5 | 0.00% | 5.94% | 1.99% | 2.16% | 0.00% |
| 20x10 | 0.03% | 15 | 0.03% | 8.77% | 3.97% | 3.68% | 0.01% |
| 20x20 | 0.02% | 27 | 0.02% | 9.46% | 3.26% | 3.06% | 0.02% |
| 50x5 | 0.03% | 25 | 0.02% | 5.10% | 0.57% | 0.64% | 0.00% |
| 50x10 | 0.82% | 94 | 0.17% | 7.04% | 4.24% | 4.25% | 0.19% |
| 50x20 | 1.29% | 164 | 0.30% | 8.78% | 5.29% | 6.15% | 0.28% |
| 100x5 | 0.06% | 70 | 0.02% | 3.57% | 0.36% | 0.36% | 0.00% |
| 100x10 | 0.55% | 162 | 0.17% | 6.92% | 1.50% | 1.72% | 0.21% |
| 100x20 | 1.96% | 830 | 0.28% | 8.28% | 4.68% | 4.81% | 0.86% |
| 200x10 | 0.49% | 861 | 0.13% | 5.60% | 0.96% | 0.89% | 0.08% |
| 200x20 | 3.04% | 881 | 0.38% | 7.60% | 4.14% | 3.65% | 2.36% |
| 500x20 | 3.05% | 3138 | 0.42% | 5.50% | 1.89% | 1.62% | 2.08% |
| avg | 0.95% | 523 | 0.16% | 6.88% | 2.74% | 2.75% | 2.74% |

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

## 6 Conclusions

The paper proposes a framework for solving combinatorial optimization problems using Apache Spark computation environment and a collective of simple optimization agents. The proposed framework denoted as MPF is flexible and can be used for solving a variety of combinatorial optimization problems. In the current paper, we demonstrate the MPF application for solving instances of job shop and flow shop scheduling problems. The idea of the MPF is based on recently proposed by the authors mushroom picking metaheuristic, where many agents explore randomly the solution space intensifying their search around promising solutions with diversification mechanism enabling escape from local optima. The approach assumes indirect cooperation between the collective members sharing access to the common memory containing a population of solutions. The computational experiment carried out, and comparisons with several recently published approaches to solving both considered scheduling problems, show that the proposed MPF implementation can obtain competitive results in a reasonable time.

Future research will focus on designing and testing a wider library of optimization agents allowing for the effortless implementation of the approach for solving a more extensive range of difficult optimization problems. Also, the framework may be extended by some new features, like for example online adjustments in the intensity of usage of the available agents. At the current version the number of agents and the frequency with which they are called is predefined. Both values could be automatically adapted during computations.

## References


1. Anuar, N.I., Fauadi, M.H.F.M., Saptari, A.: Performance Evaluation of Continuous and Discrete Particle Swarm Optimization in Job-Shop Scheduling Problems. In: Materials Science and Engineering Conference Series. Materials Science and Engineering Conference Series, vol. 530, p. 012044 (Jun 2019). https://doi.org/10.1088/1757-899X/530/1/012044
2. Balande, U., Shrimankar, D.: A modified teaching learning metaheuristic algorithm with opposite-based learning for permutation flow-shop scheduling problem. Evolutionary Intelligence pp. 1–23 (2020). https://doi.org/10.1007/s12065-020-00487-5
3. Barbucha, D., Czarnowski, I., Jedrzejowicz, P., Ratajczak, E., Wierzbowska, I.: Jade-based a-team as a tool for implementing population-based algorithms. In: Sixth International Conference on Intelligent Systems Design and Applications. vol. 3, pp. 144–149 (2006). https://doi.org/10.1109/ISDA.2006.31
4. Belabid, J., Aqil, S., Allali, K.: Solving permutation flow shop scheduling problem with sequence-independent setup time. Journal of Applied Mathematics **2020**, 1–11 (01 2020). https://doi.org/10.1155/2020/7132469
5. Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling computer and manufacturing processes (1996). https://doi.org/10.1007/978-3-662-03217-6
6. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Information Sciences **237**, 82 – 117 (2013). https://doi.org/10.1016/j.ins.2013.02.041
7. Chen, X., Zhang, B., Gao, D.: Algorithm based on improved genetic algorithm for job shop scheduling problem. pp. 951–956 (08 2019). https://doi.org/10.1109/ICMA.2019.8816334


ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

8. Danovaro, E., Clematis, A., Galizia, A., Ripepi, G., Quarati, A., D'Agostino, D.: Heterogeneous architectures for computational intensive applications: A cost-effectiveness analysis. Journal of Computational and Applied Mathematics **270**, 63 – 77 (2014). https://doi.org/10.1016/j.cam.2014.02.022
9. Dodu, C., Ancau, M.: "a tabu search approach for permutation flow shop scheduling ". Studia Universitatis Babeș-Bolyai Informatica **65**, 104–115 (07 2020). https://doi.org/10.24193/subbi.2020.1.08
10. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976), http://www.jstor.org/stable/3689278
11. Hu, H., Lei, W., Gao, X., Zhang, Y.: Job-shop scheduling problem based on improved cuckoo search algorithm. International Journal of Simulation Modelling **17**, 337–346 (06 2018). https://doi.org/10.2507/IJSIMM17(2)CO8
12. Jedrzejowicz, P.: Current trends in the population-based optimization. In: Nguyen, N.T., Chbeir, R., Exposito, E., Aniorté, P., Trawiński, B. (eds.) Computational Collective Intelligence. pp. 523–534. Springer International Publishing, Cham (2019)
13. Jedrzejowicz, P., Wierzbowska, I.: Parallelized swarm intelligence approach for solving tsp and jssp problems. Algorithms **13**(6), 142 (Jun 2020). https://doi.org/10.3390/a13060142
14. Kalshetty, Y., Adamuthe, A., Kumar, S.: Genetic algorithms with feasible operators for solving job shop scheduling problem. Journal of scientific research **64**, 310–321 (01 2020). https://doi.org/10.37398/JSR.2020.640157
15. Lawrence, S.: Resource constrained project scheduling - technical report (1984)
16. Lenstra, J., Rinnooy Kan, A., Brucker, P.: Complexity of machine scheduling problems. Annals of Discrete Mathematics **1**, 343–362 (1977). https://doi.org/10.1016/S0167-5060(08)70743-X
17. Liu, W., Jin, Y., Price, M.: A new improved neh heuristic for permutation flowshop scheduling problems. International Journal of Production Economics **193**, 21 – 30 (2017). https://doi.org/10.1016/j.ijpe.2017.06.026
18. Lopes Silva, M.A., de Souza, S.R., Freitas Souza, M.J., Bazzan, A.L.C.: A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. Expert Systems with Applications **131**, 148 – 171 (2019). https://doi.org/10.1016/j.eswa.2019.04.056
19. Lopes Silva, M.A., de Souza, S.R., Freitas Souza, M.J., de França Filho, M.F.: Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. Applied Soft Computing **71**, 433 – 459 (2018). https://doi.org/10.1016/j.asoc.2018.06.050
20. Ma, X., Li, X., Zhang, Q., Tang, K., Liang, Z., Xie, W., Zhu, Z.: A survey on cooperative co-evolutionary algorithms. IEEE Transactions on Evolutionary Computation **23**(3), 421–441 (2019). https://doi.org/10.1109/TEVC.2018.2868770
21. Pongchairerks, P.: An enhanced two-level metaheuristic algorithm with adaptive hybrid neighborhood structures for the job-shop scheduling problem. Complexity **2020**, 1–15 (2020). https://doi.org/10.1155/2020/3489209
22. Salloum, S., Dautov, R., Chen, X., Peng, P., Huang, J.: Big data analytics on apache spark. International Journal of Data Science and Analytics **1** (10 2016). https://doi.org/10.1007/s41060-016-0027-9
23. Sharma, S., Jeet, K., Nailwal, K., Gupta, D.: An improvement heuristic for permutation flow shop scheduling. International Journal of Process Management and Benchmarking **9**, 124 (01 2019). https://doi.org/10.1504/IJPMB.2019.10019077


ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34

24. Soltysova, Z., Semanco, P., Modrak, J.: Exploring heuristic techniques for flow shop scheduling. Management and Production Engineering Review **vol. 10**(No 3). https://doi.org/10.24425/mper.2019.129598
25. Sun, L., Lin, L., Li, H., Gen, M.: Large scale flexible scheduling optimization by a distributed evolutionary algorithm. Computers and Industrial Engineering **128**, 894 – 904 (2019). https://doi.org/10.1016/j.cie.2018.09.025
26. Éric Taillard: Summary of best known lower and upper bounds of Taillard's instances. http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html (2015), [Online; accessed 23-November-2020]
27. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278 – 285 (1993). https://doi.org/10.1016/0377-2217(93)90182-M, project Management anf Scheduling
28. Tsai, C.W., Chang, H.C., Hu, K.C., Chiang, M.C.: Parallel coral reef algorithm for solving jsp on spark. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016, Budapest, Hungary, October 9-12, 2016. pp. 1872–1877. IEEE (2016). https://doi.org/10.1109/SMC.2016.7844511
29. Çaliş Uslu, B., Bulkan, S.: A research survey: review of ai solution strategies of job shop scheduling problem. Journal of Intelligent Manufacturing **26** (01 2013). https://doi.org/10.1007/s10845-013-0837-8
30. Viana, M.S., Morandin Junior, O., Contreras, R.C.: A modified genetic algorithm with local search strategies and multi-crossover operator for job shop scheduling problem. Sensors **20**(18), 5440 (Sep 2020). https://doi.org/10.3390/s20185440
31. Vital-Soto, A., Azab, A., Baki, M.F.: Mathematical modeling and a hybridized bacterial foraging optimization algorithm for the flexible job-shop scheduling problem with sequencing flexibility. Journal of Manufacturing Systems **54**, 74 – 93 (2020). https://doi.org/10.1016/j.jmsy.2019.11.010
32. Wang, F., Tian, Y., Wang, X.: A discrete wolf pack algorithm for job shop scheduling problem. In: Proceedings of the 2019 5th International Conference on Control, Automation and Robotics (ICCAR), Beijing, China, pp. 19–22 (Apr 2019)
33. Wu, J.Y., Wu, X., Lu, X.Q., Du, Y.C., Zhang, M.X.: Water wave optimization for flow-shop scheduling. In: Huang, D.S., Huang, Z.K., Hussain, A. (eds.) Intelligent Computing Methodologies. pp. 771–783. Springer International Publishing (2019)
34. Yang, L., Wang, C., Gao, L., Song, Y., Li, X.: An improved simulated annealing algorithm based on residual network for permutation flow shop scheduling. Complex & Intelligent Systems pp. 1–11 (2020). https://doi.org/10.1007/s40747-020-00205-9
35. Zhang, L., Yu, Y., Luo, Y., Zhang, S.: Improved cuckoo search algorithm and its application to permutation flow shop scheduling problem. Journal of Algorithms & Computational Technology **14**, 1748302620962403 (2020). https://doi.org/10.1177/1748302620962403
36. Zhang, Z., Guan, Z., Zhang, J., Xie, X.: A novel job-shop scheduling strategy based on particle swarm optimization and neural network. International Journal of Simulation Modelling **18**, 699–707 (12 2019). https://doi.org/10.2507/IJSIMM18(4)CO18
37. Zhu, J., Shao, Z., Chen, C.: An improved whale optimization algorithm for job-shop scheduling based on quantum computing. International Journal of Simulation Modelling **18**, 521–530 (09 2019). https://doi.org/10.2507/IJSIMM18(3)CO13

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77967-2_34