An Effective Hybrid Imperialist Competitive Algorithm and Tabu Search for an Extended Flexible Job Shop Scheduling Problem

Willian Tessaro Lunardi*
University of Luxembourg
Luxembourg, Luxembourg
willian.tessarolunardi@uni.lu

Holger Voos*†
University of Luxembourg
Luxembourg, Luxembourg
holger.voos@uni.lu

Luiz Henrique Cherri
University of São Paulo
São Paulo, Brazil
lhcherri@icmc.usp.br

ABSTRACT

An extended version of the flexible job shop problem is tackled in this work. The investigated extension of the classical flexible job shop problem allows the precedences between the operations to be given by an arbitrary directed acyclic graph instead of a linear order. The problem consists of designating the operations to the machines and sequencing them in compliance with the supplied precedences. The goal in the present work is the minimization of the makespan. In order to produce reasonable outcomes in acceptable time, a hybrid imperialist competitive algorithm and tabu search is proposed to solve the problem. Numerical experiments assess the efficiency of the proposed method and compare it with well-known scheduling algorithms.

CCS CONCEPTS

• **Theory of computation → Scheduling algorithms**; *Evolutionary algorithms*; *Tabu search*;

KEYWORDS

Flexible job shop scheduling, imperialist competitive algorithm, tabu search, parallel operations

ACM Reference Format:
Willian Tessaro Lunardi, Holger Voos*, and Luiz Henrique Cherri. 2019. An Effective Hybrid Imperialist Competitive Algorithm and Tabu Search for an Extended Flexible Job Shop Scheduling Problem. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3297280.3297302

1 INTRODUCTION

The traditional job shop (JS) problem consists of scheduling n jobs on an environment with m machines. Each job is composed of several operations with a linear precedence structure and has a predetermined route through the machines. The flexible job shop scheduling (FJS) problem is a generalization of the JS problem in which there may be several machines, not necessarily identical, capable of processing each operation. The processing time of each operation on each machine is known and no preemption is allowed. The objective is to decide on which machine each operation will be processed, and in what order the operations will be processed on each machine so that a certain criterion is optimized [5].

This paper studies the extended version of the FJS problem that admits precedences between the operations to be given by an arbitrary directed acyclic graph instead of a linear order. Therefore, the problem consists of allocating the operations to the machines and sequencing them in compliance with all given precedences. An example of a type of job with this general type of precedences is presented in Figure 1. This problem appears in practical and industrial environments (e.g. printing industry, glass industry, commercial aviation, project management, etc.), where assembling and disassembling operations are part of the production process. For example, in an actual problem from the printing industry [25] some jobs consist of two independent sequences of operations followed by an assembling operation that joins the previously processed components.

In terms of computational complexity, the extended FJS problem is NP-hard, considering it is known that the JS problem is NP-hard [10], and recognized as one of the most difficult combinatorial optimization problems [14].

Several methods have been presented to solve the FJS problem. Exact methods can be used to solve small instance [4, 8, 20]. Though, once the number of jobs rises, it is difficult to find an optimal solution in a short time. Many researchers proposed heuristic methods to solve the FJS problem, such as genetic algorithm (GA) [12, 13, 16, 23, 26], particle swarm optimization (PSO) [19, 27], and tabu search (TS) [17, 22].

A few scientific works, inspired in practical applications, deal with the extension of the FJS in which precedences are given by an arbitrary directed acyclic graph. Birgin et al. [4] introduced a mixed integer linear programming (MILP) formulation and compared with the usual MILP formulations for the classical FJS problem. Moreover, a considerable set of problem instances composed of G-job and Y-job types are presented. Alvarez-Valdés et al. [1] describe an environment coming from a glass factory, that requires an even more general variant of the FJS problem including, for example, no-wait constraints. Vilcot and Billaut [21] tackle a class of instances that includes arbitrary precedence relations among operations (with the constraint of having an ending assembling operation in each job) of a problem from the printing and boarding industry. Considering the makespan and the maximum lateness criteria, TS and GA are applied with the aim of building an approximation of the Pareto frontier.

*Also with Interdisciplinary Centre for Security, Reliability and Trust (SnT).
†Also with Interdisciplinary Centre for Security, Reliability and Trust (SnT).

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/10.1145/3297280.3297302

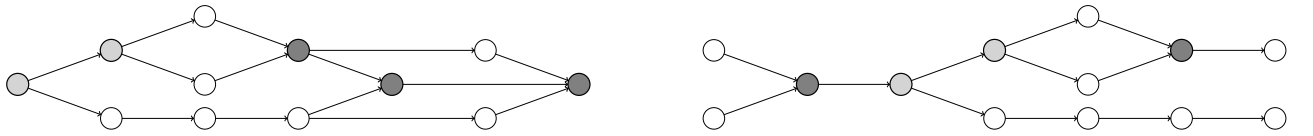


Figure 1: Two examples of job topologies that may appear in the EFJS problem. Precedence between operations are given by an arbitrary directed acyclic graph.

Due to the limited amount of works that address the extended version of the FJS problem and its practical applicability, the purpose of this paper is to contribute to the development of metaheuristics techniques able to produce reasonable results in acceptable time. A novel representation scheme is proposed in order to improve the searching capabilities of the algorithm. In order to explore the order in which parallel operations of a job are processed, a modified graph transversal algorithm together with a string that denotes cost weights for operations in the job route are proposed. A hybrid imperialist competitive algorithm and tabu search, so-called ICA+TS, is proposed to find "good enough" solutions in an acceptable time for large-sized problem instances. In order to conduct numerical experiments concerning the proposed method, we further implemented our representation scheme with a genetic algorithm (GA), and a grey wolf optimizer (GWO). Using 50 extended FJS problem instances the proposed method is examined and compared with other exact and heuristic methods, commonly employed for scheduling problems.

2 EXTENDED FLEXIBLE JOB SHOP PROBLEM

An accurate description of the considered problem can be provided by the MILP formulation introduced in [4], that we reproduce in this section for completeness and to introduce the notation that will be used in the present work.

Let n, o, and m be the number of jobs, operations, and machines, respectively. Let V be the set of all operations. For each operation $i(i = 1, ..., o)$, let $F_i \subseteq (1, 2, ..., m)$, where $F_i \neq \emptyset$, be the subset of machines that can process operation i and let $p_{ik}(i = 1, ..., o, k \in F_i)$ be the corresponding processing times. Furthermore, let A be a set of pairs (i, j) with $i, j \in \{1, ..., o\}$ such that, if (i, j) belongs to A, this means that operation i precedes operation j, i.e. operation j cannot start to be processed until operation i ends to be processed $(s_i + p_{ik} \leq s_j)$. This set of precedences A is the place where jobs are implicitly defined. The problem consists in assigning each operation i to a machine $k \in F_i$ and to determine a starting processing time s_i such that precedences are satisfied. A machine can not process more than an operation at a time and preemption is not allowed. The objective is to minimize the makespan, i.e. the completion time of the last operation.

The model uses binary variables $x_{ik}(i = 1, ..., o, k \in F_i)$ to indicate whether operation i is allocated to be processed by machine k (in this case $x_{ik} = 1$) or not (in this case $x_{ik} = 0$). It also considers binary variables $y_{ij}(i, j = 1, ..., o, F_i \cap F_j \neq \emptyset)$ to indicate, whenever two operations are allocated to the same machine, which one is processed first. Finally, the model uses variables $s_i(i = 1, ..., o)$, to denote the starting time of operation i (on the machine to which it was allocated) and a variable C_{max} to represent the makespan. With these variables, the MILP model of the extended flexible job shop problem presented in [4] can be written as:

$$\text{Minimize } C_{max} \tag{1}$$

$$\text{subject to}$$

$$\sum_{k \in F_i} x_{ik} = 1 \qquad \forall i \in V, \tag{2}$$

$$p'_i = \sum_{k \in F_i} x_{ik} p_{ik} \qquad \forall i \in V, \tag{3}$$

$$s_i + p'_i \leq C_{max} \qquad \forall i \in V, \tag{4}$$

$$s_i + p'_i \leq s_j \qquad \forall (i, j) \in A, \tag{5}$$

$$y_{ij} + y_{ji} \geq x_{ik} + x_{jk} - 1 \qquad \forall i, j \in V, i \neq j, k \in F_i \cap F_j, \tag{6}$$

$$s_i + p'_i - (1 - y_{iw})L \leq s_j \qquad \forall i, j \in V, i \neq j, F_i \cap F_j \neq \emptyset, \tag{7}$$

$$s_i \geq 0 \qquad \forall i \in V. \tag{8}$$

Constraint (2) states that each operation i must be specified to exactly one machine $k \in F_i$. Constraint (3) determines the actual processing time p'_i of each operation i (on the machine it was assigned). Constraint (4), together with the minimization of the objective function in (1) represents C_{max} as the makespan. Constraint (5) denotes the precedence constraints. For every pair of operations assigned to the same machine, constraints (6, 7) state that both operations cannot be processed at the same time and determine which one is processed first. If two operations i and j that could have been both assigned to a machine k (i.e. $k \in F_i \cap F_j \neq \emptyset$) are not then we have that at most one between x_{ik} and x_{jk} is equal to 1. In this case, constraints (6, 7) are trivially satisfied with $y_{ij} = y_{ji} = 0$. In (7), L represents a sufficiently large positive constant (see [2] for a suggested value that may be used in practice). Finally, constraint (8) states that the starting times of the operations must be not smaller than the beginning of the considered planning horizon that, without loss of generality, was set to 0.

Model (1-8) was introduced in [4], where a comparison with a simple extension of the model presented in [20] was given. Up to the authors acknowledge these two models are the only published ones that include the possibility of the precedence between the operations to be given by an arbitrary directed acyclic graph.

3 REAL-VALUED REPRESENTATION SCHEME

According to Gao et al. [9], a discrete-valued representation scheme can be used to code the FJS problem solution. The scheme is composed of two strings, one for the machine assignment problem and the other for the sequencing problem. The strings are called machine assignment string (MS) and operation sequence string (OS) and respectively referred to as ϑ_1 and ϑ_2 in this paper.

Many of the well-known metaheuristics originally work on continuous spaces because these can be formulated naturally in a real domain, e.g., PSO, GWO, ICA, magnetic optimization algorithm

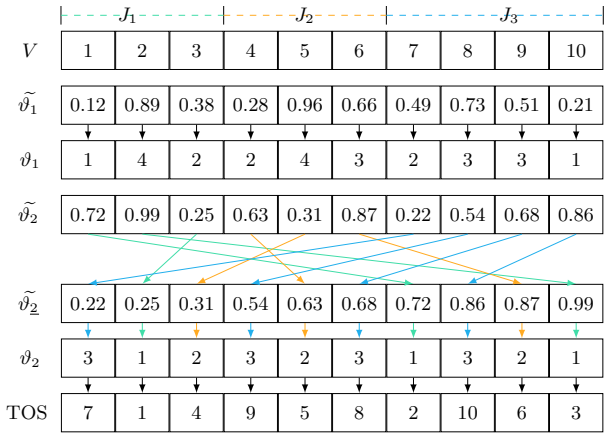


Figure 2: A random example of a representation scheme of an extended FJS problem instance containing 3 jobs, 10 operations, where every operation can be performed by all 4 machines.

(MOA), cuckoo search (CS), firefly algorithm (FA), galaxy-based search (GS), earthworm optimization (EW), lightning search (LS), moth-flame optimization (MF), sine cosine (SC), dragonfly algorithm (DA), whale optimization algorithm (WOA), and black hole (BH). Consequently, in order to use these techniques to solve the extended FJS problem, it is necessary to define a real-valued representation scheme.

Let $\vartheta_1 = \{\vartheta_{11}, \vartheta_{12}, ..., \vartheta_{1T}\}$ represent the discrete MS string where $T = \sum_{i=1}^{n} S_i$ is the problem dimension (i.e., total number of operations), and S_i is the number of operations of job i. The ith element ϑ_{1i} denotes the assigned machine for the ith operation, $\vartheta_{1i} \in F_i$. The index does not vary throughout the whole searching process. Let $\tilde{\vartheta}_1 = \{\tilde{\vartheta}_{11}, \tilde{\vartheta}_{12}, ..., \tilde{\vartheta}_{1T}\}$, similar to ϑ_1 in discrete structure, represents the continuous MS string where $\tilde{\vartheta}_{1i} \in [0, 1)$. The assigned machine for the ith operations given by $\tilde{\vartheta}_{1i}$ can be calculated as

$$\vartheta_{1i} = \left\lfloor |F_i| \; \tilde{\vartheta}_{1i} + 1 \right\rfloor, \quad i = 1, ..., T, \tag{9}$$

where $\lfloor x \rfloor$ indicates the greatest integer number smaller than x. Equation (9) can be seen represented by the arrows between every node of $\tilde{\vartheta}_1$ and ϑ_1 on Figure 2.

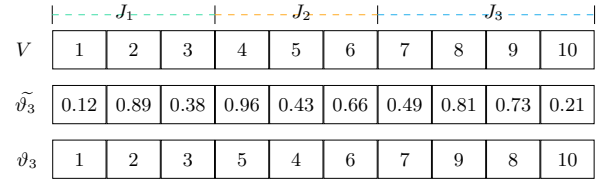


Figure 3: A random example of the parallel cost string and the MBFS outcome. The operations precedence (digraph) of the jobs used in this illustration are the following: $G_1 = 1 \rightarrow 2, 2 \rightarrow 3$, $G_2 = 4 \rightarrow 6, 5 \rightarrow 6$, and $G_3 = 7 \rightarrow 8, 7 \rightarrow 9, 8 \rightarrow 10$, where $v \rightarrow u$ means that v precedes u.

Algorithm 1 Modified Traversing Graph MBFS

```
for each job J_i do
    G_i                                   ▷ G_i: digraph of job J_i
    while ∃v ∈ G_i : v ∉ ϑ_3 do
        for each v ∈ G_i do
            if P_v ⊆ ϑ_3 ∨ P_v = ∅ then   ▷ P_v: predecessor of v
            Add v to 𝒱 end if             ▷ 𝒱: vertices to add to ϑ_3
        end for
        Sort 𝒱 in ascending order based on costs (c_i)
        for each v ∈ 𝒱 do
            {v} ∩ v_3 and {v} \ 𝒱
        end for
    end while
end for
```

Let $\vartheta_2 = \{\vartheta_{21}, \vartheta_{22}, ..., \vartheta_{2T}\}$ represent the OS string where T is the problem dimension. The string ϑ_2 denotes the order in which the operations of jobs are sequenced in their designated machines. To avoid repair mechanisms, this representation uses an unpartitioned permutation with S_i repetitions of the job numbers, i.e., every job J_i appear S_i times in ϑ_2. Let $\tilde{\vartheta}_2 = \{\tilde{\vartheta}_{21}, \tilde{\vartheta}_{22}, ..., \tilde{\vartheta}_{2T}\}$ represent the continuous OS string, where $\tilde{\vartheta}_{2i} \in [0, 1]$. In order to define a sequence of jobs as ϑ_2, the elements of $\tilde{\vartheta}_2$ are sorted (ascending or descending) into $\tilde{\vartheta}_{\underline{2}}$. In this way, the operations sequence can be defined as

$$\vartheta_{2i} = h(\tilde{\vartheta}_{\underline{2}i}), \quad i = 1, ..., T, \tag{10}$$

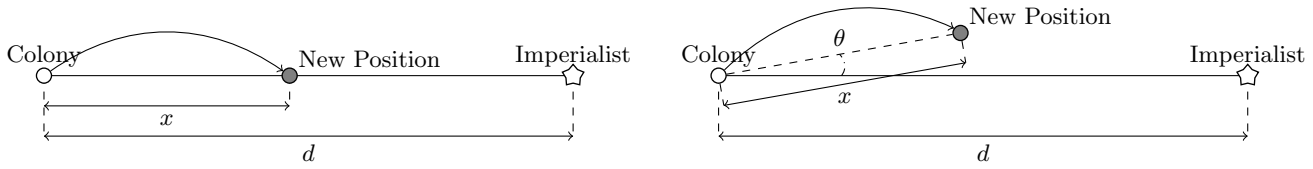
where h is a function that searches for the first appearance of the value $\tilde{\vartheta}_{\underline{2}i}$ in $\tilde{\vartheta}_2$ and gives the job number of the operation represented by the found position. Equation (10) can be seen denoted by the colored arrows between nodes of $\tilde{\vartheta}_2$, $\tilde{\vartheta}_{\underline{2}}$ and ϑ_2 in Figure 2.

At the time that ϑ_1 and ϑ_2 are defined, commonly a translation mechanism is used to replace the job numbers by its respective operations (based on topological order of the operations), e.g., based on job 3 shown in Figure 2, and a linear sequence of operations precedence, the first appearance of number 3 in ϑ_2 would be replaced by operation 7, the second appearance by the operation 8, and so on.

3.1 Extended Representation Scheme

Due to the arrangement of the operations in the job routes found in the extended FJS problem, a topological sorting algorithm can be used to define the topological order of operations of a job. The topological order is an essential feature for the translation of the OS string into a feasible sequence that respects the precedence relationship of all operations of a job. There exist well-known linear time algorithms for determining the topological order of a directed graph (digraph), e.g., Cormen [7] applies a depth-first search (DFS) algorithm. Nevertheless, with such method, the order in which parallel operations of a job appear in the generated sequence is strictly related to the policy of the method used to define it.

We extend the traditional representation scheme attaching a string to defines weights for parallel operations. The topological order of the operations of each job is defined based on the weights and a modified Breadth-First search (BFS) graph traversing algorithm. Let $\tilde{\vartheta}_3 = \{\vartheta_{31}, \vartheta_{32}, ..., \vartheta_{3T}\}$ be the continuous operation cost



(a) Assimilation operator. **(b)** Revolution operator.

Figure 4: Operators applied to evolve the colonies towards their respective imperialist.

string (CS), where $\tilde{\vartheta_{3i}} \in [0, 1]$ denotes a cost c_i for the ith operation. In this way, for any two parallel operations in the job route, the one with less cost comes before in the topological order.

A modified cost-based version of the BFS algorithm is used to define the topological order of each job. In our modified traversing graph algorithm (MBFS), a set $\mathcal{V}$ of vertices to be visited is defined. The vertices included in $\mathcal{V}$ are those where all its predecessors were already added to the topological sequence. Every vertex in $\mathcal{V}$ is visited in an ascending order based on the cost defined by $\widetilde{\vartheta_3}$ and added to ϑ_3. The pseudocode presented in Algorithm 1 illustrates the essential steps of the MBFS algorithm. Figure 3 shows a random CS string $\widetilde{\vartheta_3}$, where ϑ_3 denotes the topological order of each job given by the MBFS algorithm.

3.2 Calculation of the Makespan

Given an admissible selection and the sequencing of operations on machines, the makespan of a solution can be calculated using directed acyclic graph (DAG) $G = (V \cup \{s, e\}, A \cup Y)$ with s and e being the source and sink node respectively, and Y being a set of pairs (i, j) with $i, j \in \{1, ..., o\}$ such that, if $(i, j) \in Y$, operation i precedes operation j in the resource sequence, i.e.,

- There is a routing arc (s, i) of length 0 if i is a starting operation; a routing arc (i, j) of length p_{ik} for all $(i, j) \in A$; a routing arc (i, e) of length p_{ik} if i is an ending operation;
- There is a resource arc (i, j) of length p_{vk} for all $(i, j) \in Y$.

For any (directed) path P in G ending at i, the length of P is at most s_i. Therefore, $s_i = s_i^*$, where s_i^* is the maximum of the lengths of all paths in G ending at i. Not surprisingly, the makespan of the tight schedule is determined by the longest path $C_{max} = s_e^*$.

4 THE PROPOSED HYBRID ALGORITHM

In this paper, the proposed hybrid combines the global search and local search (LS) by using ICA to perform exploration and a guided LS algorithm to perform exploitation. Tabu search (TS) has been adopted as LS strategy. The ICA has powerful global searching ability and TS has valuable local searching ability. This method combines the advantages of ICA and TS together, being able to balance the intensification and diversification. The workflow of proposed ICA+TS is shown in Figure 5. All the details of the proposed method will be given separately in the following subsections. The overall procedure of the proposed approach is described as follows:

Step 1: Initialize the population randomly and evaluate;
Step 2: Generate initial empires;
Step 3: Is the termination criteria satisfied?
If yes, go step 10;
Else, go step 4;
Step 4: Assimilate, revolve and evaluate colonies;
Step 5: Exchange best colonies with their imperialist;
Step 6: For each imperialist, has it been exchanged in this iteration?
If yes, perform LS at imperialist solution;
Else, pick one of its colony at random and perform LS. Exchange the colony with the imperialist if it became better;
Step 7: Compute total cost of empires;
Step 8: Perform imperialist competition;
Step 9: Eliminate powerless empires and go step 3;
Step 10: Output the best solution.

4.1 Imperialist Competitive Algorithm

Encouraged by the human socio-political evolution process, a recently developed evolutionary algorithm denominated Imperialist competitive algorithm (ICA) is proposed by Atashpaz-Gargari and Lucas [3]. Similar to other evolutionary algorithms, the ICA begins with an initial population (countries). The best countries are selected to be the *imperialists*. Every imperialist has its personal set of *colonies* and the amount of colonies of each empire is equivalent to its relative power to other imperialists. The colonies of an empire evolve through operators as *assimilation* and *revolution* using the imperialist as an evolution target. Empires try to possess colonies of other empires through the *imperialistic competition* strategy. The main ICA concepts are illustrated in detail below.

4.1.1 Initial Countries. δ solutions (i.e. country) are generated randomly, where each country can be defined in form of an array

$$\zeta_i = [p_1, p_2, ..., p_\eta], \tag{11}$$

where ζ_i represents the ith country, p_i the variables, and η the total number of variables, i.e. η-dimensions of the problem to be optimized.

4.1.2 Initial Empires. α best countries are elected to be imperialist. The remaining $\delta - \alpha$ countries are used to create the colony set of each imperialist. The number of colonies that each imperialist possesses is proportional to its relative power to other imperialists defined as

$$P_k = \left| \frac{Y_k}{\sum_{i=1}^{\alpha} Y_i} \right|, \tag{12}$$

where P_k is the power of the imperialist k, $Y_i = max_k\{c_k\} - c_i$ indicates the normalized cost, and c_k is the cost of the kth imperialist. The number of initial colonies possessed by imperialist k is

calculated as round($P_k \times (\delta - \alpha)$), where round is a function that gives the nearest integer of a fractional number.

4.1.3 Assimilation and Revolution. Assimilation leads colonies to have similar features with their corresponding imperialist. During the assimilation strategy, colonies move

$$x \sim U[0, \beta d] \tag{13}$$

units towards their relevant imperialist, where x is a random value generated using uniform distribution, β is the assimilation factor and d is the distance between the colony and the imperialist. Figure 4a shows the movement of a colony towards its imperialist.

Comparable to the mutation operator in GA, the revolution operator is added to ICA to improve the exploration property of the algorithm. In order to search nearby the imperialist, a random amount of deviation

$$\theta \sim U[-y, y] \tag{14}$$

is incorporated in the movement, where θ is a random value generated using uniform distribution, y is a parameter that adjusts the deviation from the original direction. Figure 4b shows the movement of a colony towards its imperialist in a randomly deviated direction.

4.1.4 Imperialist Exchange. If a colony is better than its imperialist due to assimilation and revolution operations, the position of the imperialist and the colony are changed, i.e., the colony becomes the imperialist and vice-versa.

4.1.5 Imperialist Competition. In this step, the weakest colony of the weakest imperialist is possessed by other stronger imperialists. This is carried out in a stochastic way. The possession probability for each imperialist is related to its total cost. The better the imperialist is, the more likely it will possess the weakest colony of the weakest empire. The total cost, which is used as a comparison criterion in this step, is defined by

$$TC_i = C_i + \xi \frac{1}{n_i^\delta} \sum_{j=1}^{n_i^\delta} C_{ij}, \tag{15}$$

where C_i and TC_i are respectively the cost and total cost values for imperialist i, n_i^δ is the number of colonies of the ith empire, C_{ij} is the cost related to jth colony of empire i, and $\xi < 1$ is the colonies consideration rate. In the imperialist competition step, if the weakest imperialist loses all of its colonies, then this imperialist is collapsed. A collapsed imperialist is possessed by other imperialists as a colony.

4.2 Tabu Search

A local search algorithm starts with some initial solution and moves from neighbor to neighbor in order to find better solutions. LS method comprises tabu search (TS), simulated annealing (SA), variable neighborhood search (VNS) and so on. Its effectiveness mainly depends on the design of neighborhood structures. The main problem with this strategy is to escape from local minima where the search cannot find any further neighborhood solution that decreases the objective function value.

Tabu search [11] has been successfully applied in various combinatorial optimization problems including several scheduling problems. TS allows the search to examine solutions that do not decrease the objective function value, only in those cases where these solutions are not forbidden. This is usually obtained by keeping track of the last solutions in terms of the action used to transform one solution to the next. When an action is performed it is considered *tabu* for the next T iterations, where T is the tabu status length. A solution is forbidden if it is obtained by applying a tabu action to the current solution. The neighborhood structure adopted in this work is proposed in [17] and it was proved to be optimum connected.

In our proposed hybrid, we apply LS on every imperialist after every time it is exchanged with another colony. In order to apply LS on an imperialist, it should be converted to a feasible schedule

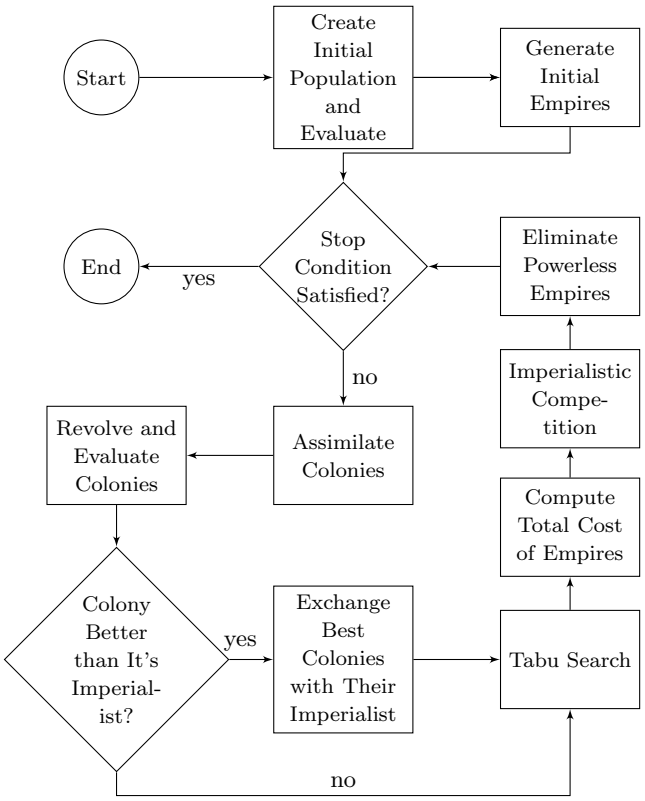


Figure 5: Operators applied to evolve the colonies towards their respective imperialist.

Table 1: The ICA+TS parameters used in the experiments.

Parameters	
Number of iterations	300
Population size, δ	500
Number of imperialist, α	10
Assimilation factor, β	2
Colonies consideration rate, ξ	0.02
TS iteration size	500 x (cur. iter. / max. iter.)
Length of tabu list	see [17] for more information

Table 2: Experiment with the YFJS instances.

Instance	CPLEX (10h limit)	GWO		GA		ICA		ICA+TS	
	C_{max}	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)
YFJS01	**773**	**773**	7.86	**773**	8.31	**773**	10.89	**773**	11.85
YFJS02	**825**	843	8.32	848	8.14	843	9.91	**825**	13.29
YFJS03	**347**	348	5.45	356	6.27	**347**	6.80	**347**	7.34
YFJS04	**390**	**390**	9.65	**390**	6.89	**390**	8.28	**390**	10.64
YFJS05	**445**	452	12.51	452	7.46	452	8.57	**445**	9.89
YFJS06	**446**	450	7.02	450	8.20	447	10.05	**446**	11.2
YFJS07	**444**	455	6.97	480	8.46	455	9.98	**444**	11.09
YFJS08	**353**	**353**	8.12	**353**	8.07	**353**	10.15	**353**	10.50
YFJS09	**242**	**242**	8.49	**242**	7.94	**242**	10.58	**242**	11.68
YFJS10	**399**	**399**	9.17	**399**	8.50	**399**	10.51	**399**	11.59
YFJS11	**526**	529	9.93	529	10.24	529	12.87	**526**	11.68
YFJS12	**512**	517	9.79	540	9.88	517	13.21	**512**	18.72
YFJS13	**405**	409	11.98	409	10.09	**405**	13.64	**405**	16.79
YFJS14	**1317**	**1317**	37.61	**1317**	40.04	**1317**	34.88	**1317**	11.75
YFJS15	[1239;1244]	1270	41.89	1269	40.13	1270	45.20	**1239**	18.42
YFJS16	[1222;1243]	1301	38.77	1301	66.90	1254	40.03	**1222**	14.30
YFJS17	[1133;1622]	1204	81.97	1204	105.49	1167	63.90	**1133**	3.93
YFJS18	[1220;2082]	1283	92.61	1283	110.50	1221	63.79	**1220**	4.39
YFJS19	[926;1525]	1153	59.19	1080	63.89	1080	53.88	**941**	10.64
YFJS20	[968;2020]	1204	59.29	1204	63.51	1079	65.16	**973**	12.29

solution at first. This solution is used as the initial solution of the TS. After the LS, the output solution of the TS should be encoded to a feasible representation scheme for the ICA's operators.

5 NUMERICAL RESULTS

This section presents the results of computational experiments involving the proposed ICA+TS algorithm. The ICA+TS performance is analyzed with other algorithms using 50 extended FJS problem instances [4]. Instances YFJS01-YFJS20 correspond to instances composed by Y-jobs; while instances DAFJS01-DAFJS30 correspond to instances composed by jobs whose precedences are given by arbitrary directed acyclic graphs (G-job).

Due to the distinctive computing hardware, programming platforms, and coding skills used to implement the proposed algorithms in the FJS problem literature, comparisons of efficiency are notoriously problematic [24]. In order to impartially analyze the effectiveness and mainly efficiency of the proposed ICA+TS, we implemented a GA, which is the most applied method for the FJS problem (see Chaudhry and Khan [6], Amjad et al. [2]), and a new meta-heuristic called Grey Wolf Optimizer (GWO) [18]. Moreover, analyzes with the ICA without LS are also presented.

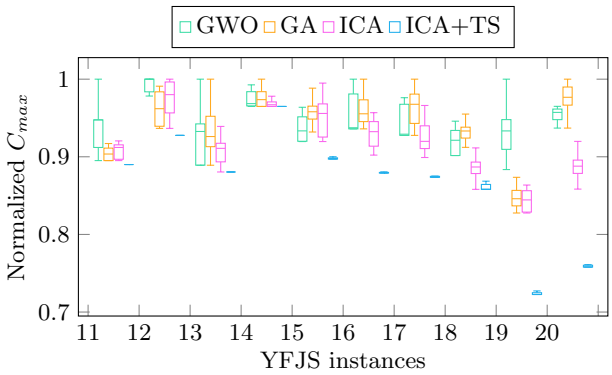


Figure 6: Distribution shape, central value, and variability of the makespan obtained with the experiments involving the YFJS instances.

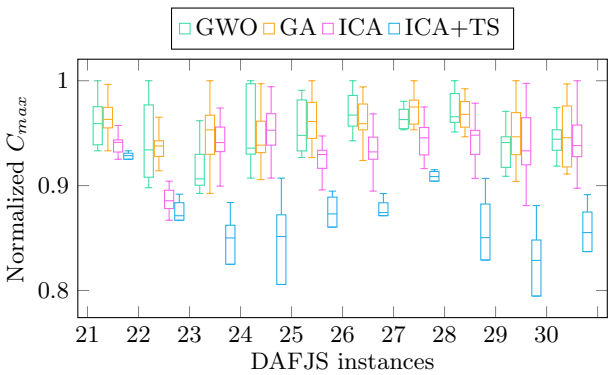


Figure 7: Distribution shape, central value, and variability of the makespan obtained with the experiments involving the DAFJS instances.

Table 3: Experiment with the DAFJS instances.

Instance	CPLEX (10h limit)	GWO		GA		ICA		ICA+TS	
	C_{max}	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)
DAFJS01	**257**	**257**	8.87	**257**	6.66	**257**	7.87	**257**	8.95
DAFJS02	**289**	**289**	8.79	**289**	6.77	**289**	7.55	**289**	8.46
DAFJS03	**576**	**576**	14.1	**576**	10.38	**576**	8.14	**576**	9.06
DAFJS04	**606**	**606**	13.4	**606**	8.96	**606**	11.47	**606**	12.17
DAFJS05	[351.57;402]	421	12.56	421	8.71	424	10.7	**389**	10.23
DAFJS06	[326;431]	414	15.84	414	9.19	423	11.26	**412**	11.66
DAFJS07	[497.90;565]	583	13.23	583	15.86	610	12.51	**512**	13.43
DAFJS08	[628;631]	655	16.65	655	16.11	642	12.0	**628**	13.51
DAFJS09	[315;484]	483	11.08	474	9.8	466	11.73	**464**	12.97
DAFJS10	[336;569]	537	10.51	537	11.78	**533**	15.37	**533**	16.15
DAFJS11	[658;708]	732	24.56	732	21.25	750	20.79	**659**	23.43
DAFJS12	[530;720]	731	15.97	731	21.97	698	24.01	**645**	26.48
DAFJS13	[304;710]	655	11.22	655	12.62	**653**	15.9	**653**	17.98
DAFJS14	[358.95;838]	737	17.87	737	14.17	735	17.17	**726**	19.97
DAFJS15	[512;818]	747	18.54	736	22.67	747	24.14	**671**	25.95
DAFJS16	[640;831]	780	20.84	778	22.63	768	34.29	**679**	36.73
DAFJS17	[300;904]	812	15.52	806	16.77	800	19.96	**787**	20.49
DAFJS18	[322;951]	799	16.14	790	15.03	790	18.46	**789**	19.95
DAFJS19	[512;595]	546	12.63	540	13.33	540	18.41	**524**	19.27
DAFJS20	[434;815]	700	19.66	700	23.41	**696**	23.41	**696**	24.72
DAFJS21	[504;965]	810	19.78	810	26.49	803	26.49	**803**	28.81
DAFJS22	[464;902]	722	20.83	722	22.46	697	29.21	**697**	32.49
DAFJS23	[450;541]	515	13.49	515	14.71	519	19.51	**476**	20.65
DAFJS24	[476;660]	635	15.0	634	17.97	635	25.0	**564**	26.02
DAFJS25	[584;897]	810	15.7	810	23.4	783	24.26	**752**	25.84
DAFJS26	[565;903]	806	26.13	790	31.51	765	22.47	**745**	24.81
DAFJS27	[503;981]	876	26.03	876	35.12	842	24.17	**831**	25.55
DAFJS28	[535;662]	623	18.47	620	22.78	594	17.25	**543**	18.48
DAFJS29	[609;720]	748	17.27	744	17.85	725	18.62	**654**	19.41
DAFJS30	[467;637]	609	17.08	604	18.76	595	19.74	**555**	20.36

The implementation of our GA is based on its classical structure (see [15]), and the operators are a random crossover and a random mutation applied to the continuous strings. Based on experiments, tournament selection produced better results than fitness proportionate selection, due to the fact that is better in maintaining the diversity of the population. Regarding the implemented GWO algorithm, it mimics the leadership hierarchy and hunting mechanism of grey wolves (Canis lupus) in nature. Four types of grey wolves such as alpha, beta, delta, and omega are employed for simulating the leadership hierarchy. The main steps of the algorithm are hunting, searching for prey, encircling prey, and attacking prey. To sum up, the search process starts with creating a random population of grey wolves (candidate solutions). Over the course of iterations, alpha, beta, and delta wolves estimate the probable position of the prey. Each candidate solution renews its distance from the prey. Some parameters are adjusted during the searching process in order to emphasize exploration and exploitation, respectively. In this way, during the first half iterations, the candidate solutions tend to diverge from the prey at first and then converge towards the prey.

The parameters of the algorithms were defined based on numerical experiments. The GA parameters defined for the experiments are 1000 generations, 500 individuals per generation, 0.9 crossover rate, and 0.3 mutation rate. The GWO has fewer parameters to be adjusted. The parameters of the implemented GWO were kept the identical as the proposed in [18] with 1000 iterations and 500 wolves per iteration.

The stated ICA+TS, GA, and GWO were implemented in C++ and run by a 2*Intel Xeon E5-2680 v3 @ 2.5 GHz with 24 CPUs on a Linux HPC cluster at the University of Luxembourg. Each experiment was performed 25 times for each algorithm. Based on investigations we find out that the best parameters for the ICA applied for the extended FJS problem are $[100, 2000]$ iterations, $\delta = [200, 500]$, and $\alpha = [5, 15]$. The ICA+TS parameters used in the experiments are shown in Table 1. Supplemental numerical results, charts, and the instances are provided at https://anonymousresearchgroup.com.

Tables 2 and 3 show the results for the two sets of extended FJS instances. The Instance column records the names of the instances. Details a number of jobs, number of operation per job and number of machines of each instances instance can be seen in [4]. Column

CPLEX shows the results obtained with the model (1-8) and the exact solver IBM ILOG CPLEX 12.1 with CPU time limit was set to 10 hours [5]. In the cases in which the exact solver achieved the CPU time limit without finding an optimal solution, it is reported the obtained lower and upper bounds. In the tables, "CPU(s)" stands for the elapsed CPU time in seconds. Column C_{max} shows the best-obtained value for each algorithm. The highlighted values (bold) denote the best-obtained values between the compared methods.

In the first set of experiments shown in Table 2, we evaluate the numerical performance of the proposed ICA+TS for instances composed of Y-jobs. The proposed algorithm obtained all the optimal values achieved by the CPLEX with CPU time smaller than 60 seconds and closed instances YFJS15 and YFJS16. For most of the Y-job instances, the ICA is more effective than the GWO and GA. The ICA+TS is more (or equal) effective than all other methods for all Y-job instances with a small loss of efficiency when compared with the ICA.

In the second set of experiments shown in Table 3, we evaluate the numerical performance of the proposed method for instances composed of G-jobs. The proposed algorithm obtained all the optimal values achieved by the exact method with CPU time smaller than 20 seconds and closed instance DAFJS08. The ICA+TS is more (or equal) effective than all other methods for all G-job instances with a small loss of efficiency when compared with the ICA.

Figures 6 and 7 gives the makespan distribution and variance for the collected numerical results over the 10 largest YFJS and DAFJS instances. It is possible to see that the proposed algorithm achieved better results with a more dense distribution and smaller makespan variance when compared to the other methods.

Here is a summary of our results. The ICA is more effective than the GWO and GA for the extended FJS problem. The proposed ICA+TS is a consistent and steady algorithm, obtained better values and lower variance in all experiments. Based on the bounds given by the CPLEX, we can see that the ICA+TS decreased the gap (i.e. the distance between lower and upper bounds) for several instances and closed instances YFJS15, YFJS16, and DAFJS08.

6 CONCLUSION

In the present paper, we extended the definition of the extended FJS problem where operations precedences of jobs are given by arbitrary directed acyclic graphs. A novel representation scheme is proposed. In this way, a string is applied to define weights for operations in the job routes and a modified graph transversal algorithm is employed to explore the order in which parallel operations are processed. We put forward a hybrid imperialist competitive algorithm and tabu search to solve the extended FJS problem. In order to evaluate the performance of the solution methods, 50 extended FJS problem instances were used. The experiments among the proposed method and others famous algorithms show that it is a feasible effective approach for the considered problem.

REFERENCES

[1] Ramón Alvarez-Valdés, A Fuertes, José Manuel Tamarit, G Giménez, and R Ramos. 2005. A heuristic to schedule flexible job-shop in a glass factory. *European journal of operational research* 165, 2 (2005), 525–534.

[2] Muhammad Kamal Amjad, Shahid Ikramullah Butt, Rubeena Kousar, Riaz Ahmad, Mujtaba Hassan Agha, Zhang Faping, Naveed Anjum, and Umer Asgher. 2018. Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems. *Mathematical Problems in Engineering* 2018 (2018).

[3] Esmaeil Atashpaz-Gargari and Caro Lucas. 2007. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In *IEEE Congress on Evolutionary computation, 2007.* 4661–4667.

[4] Ernesto G Birgin, Paulo Feofiloff, Cristina G Fernandes, Everton L De Melo, Marcio TI Oshiro, and Débora P Ronconi. 2014. A MILP model for an extended version of the Flexible Job Shop Problem. *Optimization Letters* 8, 4 (2014), 1417–1431.

[5] Ernesto G Birgin, JE Ferreira, and DP Ronconi. 2014. List scheduling and beam search methods for an extended version of the flexible job shop scheduling problem. (2014).

[6] Imran Ali Chaudhry and Abid Ali Khan. 2016. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research* 23, 3 (2016), 551–591.

[7] Thomas H Cormen. 2009. *Introduction to algorithms.* MIT press.

[8] Parviz Fattahi, Mohammad Saidi Mehrabad, and Fariborz Jolai. 2007. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of intelligent manufacturing* 18, 3 (2007), 331.

[9] Jie Gao, Mitsuo Gen, and Linyan Sun. 2006. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing* 17, 4 (2006), 493–507.

[10] Michael R Garey, David S Johnson, and Ravi Sethi. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research* 1, 2 (1976), 117–129.

[11] Fred Glover and Manuel Laguna. 1998. Tabu search. In *Handbook of combinatorial optimization.* Springer, 2093–2229.

[12] Nhu Binh Ho, Joc Cing Tay, and Edmund M-K Lai. 2007. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research* 179, 2 (2007), 316–333.

[13] Mikkel T Jensen. 2003. Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on evolutionary computation* 7, 3 (2003), 275–288.

[14] Eugene L Lawler, Jan Karel Lenstra, Alexander HG Rinnooy Kan, and David B Shmoys. 1993. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science* 4 (1993), 445–522.

[15] Xinyu Li and Liang Gao. 2016. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics* 174 (2016), 93–110.

[16] Willian Tessaro Lunardi and Holger Voos. 2018. Comparative Study of Genetic and Discrete Firefly Algorithm for Combinatorial Optimization. In *33rd Symposium On Applied Computing (SAC), 2018.* ACM/SIGAPP, 1–8.

[17] Monaldo Mastrolilli and Luca Maria Gambardella. 2000. Effective neighbourhood functions for the flexible job shop problem. *Journal of scheduling* 3, 1 (2000), 3–20.

[18] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. 2014. Grey wolf optimizer. *Advances in engineering software* 69 (2014), 46–61.

[19] Ghasem Moslehi and Mehdi Mahnam. 2011. A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics* 129, 1 (2011), 14–22.

[20] Cemal Özgüven, Lale Özbakır, and Yasemin Yavuz. 2010. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling* 34, 6 (2010), 1539–1548.

[21] Geoffrey Vilcot and Jean-Charles Billaut. 2008. A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem. *European Journal of Operational Research* 190, 2 (2008), 398–411.

[22] Geoffrey Vilcot and Jean-Charles Billaut. 2011. A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. *International Journal of Production Research* 49, 23 (2011), 6963–6980.

[23] Xiaojuan Wang, Liang Gao, Chaoyong Zhang, and Xinyu Shao. 2010. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* 51, 5 (2010), 757–767.

[24] Yuan Yuan, Hua Xu, and Jiadong Yang. 2013. A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing* 13, 7 (2013), 3259–3272.

[25] Jun Zeng, I-Jong Lin, Gary Dispoto, Eric Hoarau, and Giordano Beretta. 2011. On-demand digital print services: a new commercial print paradigm as an it service vertical. In *SRII Global Conference (SRII), 2011 Annual.* IEEE, 120–125.

[26] Guohui Zhang, Liang Gao, and Yang Shi. 2011. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications* 38, 4 (2011), 3563–3573.

[27] Guohui Zhang, Xinyu Shao, Peigen Li, and Liang Gao. 2009. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering* 56, 4 (2009), 1309–1318.