Hybrid metaheuristics within a holonic multiagent model for the flexible job shop problem

Houssem Eddine Nouri\textsuperscript{a, *}, Olfa Belkahla Drissa\textsuperscript{a,b}, Khaled Ghédira\textsuperscript{a}

\textsuperscript{a}Stratégies d’Optimisation et Informatique intelligentE, Higher Institute of Management of Tunis, Bardo, Tunis, Tunisia

\textsuperscript{b}Higher Business School of Tunis, University of Manouba, Tunisia

Abstract

The Flexible Job Shop scheduling Problem (FJSP) is an extension of the classical Job Shop scheduling Problem (JSP) that allows to process operations on one machine out of a set of alternative machines. It is an NP-hard problem consisting of two sub-problems which are the assignment and the scheduling problems. This paper proposes a hybridization of two metaheuristics within a holonic multiagent model for the FJSP. Firstly, a scheduler agent applies a Neighborhood-based Genetic Algorithm (NGA) for a global exploration of the search space. Secondly, a cluster agents set uses a local search technique to guide the research in promising regions. Numerical tests are made to evaluate our approach, based on two sets of benchmark instances from the literature of the FJSP, which are the Brandimarte and Hurink data. The experimental results show the efficiency of our approach in comparison to other approaches.

1. Introduction

The Job Shop scheduling Problem (JSP), which is among the hardest combinatorial optimization problems\textsuperscript{1}, is a branch of the industrial production scheduling problems. The Flexible Job Shop scheduling Problem (FJSP) is an extension of the classical JSP, allowing to process operations on one machine out of a set of alternative machines. Hence, the FJSP is more computationally difficult than the JSP. Furthermore the operation scheduling problem, the FJSP presents an additional difficulty caused by the operation assignment problem to a set of available machines. This problem is known to be strongly NP-Hard even if each job has at most three operations and there are two machines\textsuperscript{2}.

To solve this problem, researchers used the metaheuristics to find near-optimal solutions for the FJSP with acceptable computational time. Brandimarte\textsuperscript{3} proposed a hierarchical algorithm based on Tabu Search metaheuristic for routing and scheduling with some known dispatching rules to solve the FJSP. Hurink\textsuperscript{4} developed a Tabu Search procedure for the job shop problem with multi-purpose machines. Mastroilli\textsuperscript{5} used Tabu Search techniques and...
presented two neighborhood functions allowing an approximate resolution for the FJSP. Bozejko\textsuperscript{6} presented a Tabu Search approach based on a new golf neighborhood for the FJSP. For the Genetic Algorithm, it was adopted by\textsuperscript{7}, where their chromosome representation of solutions for the problem was divided into two parts. The first part defined the routing policy and the second part took the sequence of operations on each machine. Gao\textsuperscript{8} adapted a hybrid Genetic Algorithm (GA) and a Variable Neighborhood Descent (VND) for FJSP. The GA used two vectors to represent a solution and the disjunctive graph to calculate it. Then, a VND was applied to improve the GA final individuals. Zhang\textsuperscript{9} presented a model of low-carbon scheduling in the FJSP considering three factors, the makespan, the machine workload for production and the carbon emission for the environmental influence. A metaheuristic hybridization algorithm was proposed combining the original Non-dominated Sorting Genetic Algorithm II (NSGA-II) with a Local Search algorithm based on a neighborhood search technique. In addition, a new heuristic was developed by\textsuperscript{10} for the FJSP. This heuristic is based on a constructive procedure considering simultaneously many factors having a great effect on the solution quality. Furthermore, distributed artificial intelligence techniques were used for this problem, such as the multiagent model proposed by\textsuperscript{11} composed by three classes of agents, job agents, resource agents and an interface agent. This model is based on a local search method which is the tabu search to solve the FJSP. Also, this model was improved in\textsuperscript{12} where the optimization role of the interface agent was distributed among the resource agents. Henchiri\textsuperscript{13} proposed a multiagent model based on a hybridization of two metaheuristics, a local optimization process using the tabu search to get a good exploitation of the good areas and a global optimization process integrating the Particle Swarm Optimization (PSO) to diversify the search towards unexplored areas.

In this paper, we propose a hybrid two metaheuristics based within a holonic multiagent model for the flexible job shop scheduling problem. This new approach follows two principal hierarchical steps, where a genetic algorithm is applied by a scheduler agent for a global exploration of the search space. Then, a local search technique is used by a set of cluster agents to guide the research in promising regions. Numerical tests were made to evaluate the performance of our approach based on two data sets of\textsuperscript{3} and\textsuperscript{4} for the FJSP, completed by comparisons with other approaches.

The rest of the paper is organized as follows. In section 2, we define the formulation of the FJSP with its objective function and a simple problem instance. Then, in section 3, we detail the proposed hybrid approach with its holonic multiagent levels. The experimental and comparison results are provided in section 4. Finally, section 5 ends the paper with a conclusion.

2. Problem formulation

The flexible job shop scheduling problem (FJSP) could be formulated as follows. There is a set of $n$ jobs $J = \{J_1, \ldots, J_n\}$ to be processed on a set of $m$ machines $M = \{M_1, \ldots, M_m\}$. Each job $J_i$ is formed by a sequence of $n_i$ operations $\{O_{i1}, O_{i2}, \ldots, O_{i,n_i}\}$ to be performed successively according to the given sequence. For each operation $O_{i,j}$, there is a set of alternative machines $M(O_{i,j})$ capable of performing it. The main objective of this problem is to find a schedule minimizing the end date of the last operation of the jobs set which is the makespan. The makespan is defined by $C_{max}$ in Equation (1), where $C_i$ is the completion time of a job $J_i$.

$$C_{max} = \max_{1\leq i \leq n}(C_i)$$

To explain the FJSP, a sample problem of three jobs and five machines is shown in Table 1, where the numbers present the processing times and the tags “–” mean that the operation cannot be executed on the corresponding machine.

3. Hybrid metaheuristics within a holonic multiagent model

Glover\textsuperscript{14} elaborated a study about the nature of connections between the genetic algorithm and tabu search metaheuristics, searching to show the existing opportunities for creating a hybrid approach with these two standard methods to take advantage of their complementary features and to solve difficult optimization problems. After this pertinent study, the combination of these two metaheuristics has become more well-known in the literature, which has motivated many researchers to try the hybridization of these two methods for the resolution of different complex problems in several areas. Ferber\textsuperscript{15} defined a multiagent system as an artificial system composed of a population of autonomous
agents, which cooperate with each other to reach common objectives, while simultaneously each agent pursues individual objectives. Furthermore, a multiagent system is a computational system where two or more agents interact (cooperate or compete, or a combination of them) to achieve some individual or collective goals. The achievement of these goals is beyond the individual capabilities and individual knowledge of each agent\textsuperscript{16}. Koestler\textsuperscript{17} gave the first definition of the term “holon” in the literature, by combining the two Greek words “hol” meaning whole and “on” meaning particle or part. He said that almost everything is both a whole and a part at the same time. In fact, a holon is recursively decomposed at a lower granularity level into a community of other holons to produce a holarchy\textsuperscript{18}. Moreover, a holon may be viewed as a sort of recursive agent, which is a super-agent composed by a sub-agents set, where each sub-agent has its own behavior as a complementary part of the whole behaviour of the super-agent. Holons are agents able to show an architectural recursiveness\textsuperscript{19}.

In this work, we propose a hybrid metaheuristic approach processing two general steps: a first step of global exploration using a genetic algorithm to find promising areas in the search space and a clustering operator allowing to regroup them in a set of clusters. In the second step, a tabu search algorithm is applied to find the best individual solution for each cluster. The global process of the proposed approach is implemented in two hierarchical holonic levels adopted by a recursive multiagent model, named hybrid Genetic Algorithm with Tabu Search within a Holonic Multiagent model (GATS+HM), see Figure 1. The first holonic level is composed by a Scheduler Agent which is the Master/Super-agent, preparing the best promising regions of the search space, and the second holonic level containing a set of Cluster Agents which are the Workers/Sub-agents, guiding the search to the global optimum solution of the

**Table 1. A simple instance of the FJSP**

| Job | Operation | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|-----|-----------|-------|-------|-------|-------|-------|
| $J_1$ | $O_{1,1}$ | 2 | 9 | 4 | 5 | 1 |
|     | $O_{1,2}$ | – | 6 | – | 4 | – |
| $J_2$ | $O_{2,1}$ | 1 | – | 5 | – | 6 |
|     | $O_{2,2}$ | 3 | 8 | 6 | – | – |
|     | $O_{2,3}$ | – | 5 | 9 | 3 | 9 |
| $J_3$ | $O_{3,1}$ | – | 6 | 6 | – | – |
|     | $O_{3,2}$ | 3 | – | – | 5 | 4 |

![Fig. 1. Hybrid metaheuristics within a holonic multiagent model](image-url)
problem. Each holonic level of this model is responsible to process a step of the hybrid metaheuristic algorithm and to cooperate between them to attain the global solution of the problem.

In fact, the choice of this new metaheuristic hybridization is justified by that the standard metaheuristic methods use generally the diversification techniques to generate and to improve many different solutions distributed in the search space, or by using local search techniques to generate a more improved set of neighbourhood solutions from an initial solution. But they did not guarantee to attain promising areas with good fitness converging to the global optimum despite the repetition of many iterations, that is why they need to be more optimized. So, the novelty of our approach is to launch a genetic algorithm based on a diversification technique to only explore the search space and to select the best promising regions by the clustering operator. Then, applying the intensification technique of the tabu search allowing to relaunch the search from an elite solution of each cluster autonomously to attain more dominant solutions of the search space. Also, the use of a multiagent system gives the opportunity for distributed and parallel treatments which are very complimentary for the second step of the proposed approach. Indeed, our combined metaheuristic approach follows the paradigm of “Master” and “Workers” which are two recursive hierarchical levels adaptable for a holonic multiagent model, where the Scheduler Agent is the Master/Super-agent of its society and the Cluster Agents are its Workers/Sub-agents.

3.1. Scheduler Agent

The Scheduler Agent (SA) is responsible to process the first step of the hybrid algorithm by using a genetic algorithm called NGA (Neighborhood-based Genetic Algorithm) to identify areas with high average fitness in the search space during a fixed number of iterations MaxIter. In fact, the goal of using the NGA is only to explore the search space, but not to find the global solution of the problem. Then, a clustering operator is integrated to divide the best identified areas by the NGA in the search space to different parts where each part is a cluster $CL_i \in CL$ the set of clusters, where $CL = \{CL_1, CL_2, \ldots, CL_N\}$. In addition, this agent plays the role of an interface between the user and the system (initial parameter inputs and final result outputs). According to the number of clusters $N$ obtained after the integration of the clustering operator, the SA creates $N$ Cluster Agents (CAs) preparing the passage to the next step of the global algorithm. After that, the SA remains in a waiting state until the reception of the best solutions found by the CA for each cluster. Finally, it finishes the process by displaying the final solution of the problem.

![Fig. 2. The chromosome representation of a scheduling solution](image)

3.1.1. Individual’s solution presentation

The flexible job shop problem is composed by two sub-problems: the machine assignment problem and the operation scheduling problem, that is why the chromosome representation is encoded in two parts: Machine Assignment part (MA) and Operation Sequence part (OS). The first part MA is a vector $V_1$ with a length $L$ equal to the total number of operation and where each index represents the selected machine to process an operation indicated at position $p$, see Figure 2 (a). For example $p = 2$, $V_1(2)$ is the selected machine $M_4$ for the operation $O_{1,2}$. The second part OS is a vector $V_2$ having the same length of $V_1$ and where each index represents an operation $O_{i,j}$ according to the predefined operations of the job set, see Figure 2 (b). For example the operation sequence $1 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 3$ can be translated to: $(O_{1,1}, M_5) \rightarrow (O_{2,1}, M_1) \rightarrow (O_{1,2}, M_4) \rightarrow (O_{3,1}, M_3) \rightarrow (O_{2,2}, M_3) \rightarrow (O_{5,2}, M_1) \rightarrow (O_{2,3}, M_2)$.

To convert the chromosome values to an active schedule, we used the priority-based decoding of \(^8\). This method considers the idle time which may exist between operations on a machine $m$, and which is caused by the precedence...
constraints of operations belonging to the same job $i$. Let $S_{i,j}$ is the starting time of an operation $O_{i,j}$ (which can only be started after processing its precedent operation $O_{i,(j-1)}$) with its completion time $C_{i,j}$. In addition, we have an execution time interval $[t^S_m, t^E_m]$ starts form $t^S_m$ and ends at $t^E_m$ on a machine $m$ to allocate an operation $O_{i,j}$. So, if $j = 1, S_{i,j}$ takes $t^S_m$, else if $j \geq 2$, it takes $\max(t^S_m, C_{i,(j-1)})$. In fact, the availability of the time interval $[t^S_m, t^E_m]$ for an operation $O_{i,j}$ is validated by verifying if there is a sufficient time period to complete the execution time $p_{j,m}$ of this operation, see Equation (2):

$$if \ j = 1, t^S_m + p_{j,m} \leq t^E_m$$

$$if \ j \geq 2, \max(t^S_m, C_{i,(j-1)}) + p_{j,m} \leq t^E_m$$

The used priority-based decoding method allows in each case to assign each operation to its reserved machine following the presented execution order of the operation sequence vector $V_2$. Also, to schedule an operation $O_{i,j}$ on a machine $m$, the fixed idle time intervals of the selected machine are verified to find an allowed available period to its execution. So, if a period is found, the operation $O_{i,j}$ is executed there, else it is moved to be executed at the end of the machine $m$. Noting that the chromosome fitness is calculated by $Fitness(i)$ which is the fitness function of each chromosome $i$ and $Cmax(i)$ is its makespan value, where $i \in \{1, \ldots, P\}$ and $P$ is the total population size, see Equation (3).

$$Fitness(i) = \frac{1}{Cmax(i)}$$

3.1.2. Population initialization

The initial population is generated randomly following a uniform law and based on a neighborhood parameter to make the individual solutions more diversified and distributed in the search space. In fact, each new solution should have a predefined distance with all the other solutions to be considered as a new member of the initial solution. The used method to determinate the neighborhood parameter is inspired from 6, which is based on the permutation level of operations to obtain the dissimilarity distance between two solutions. Let $Chrom1(MA_1, OS_1)$ and $Chrom2(MA_2, OS_2)$ two chromosomes of two different scheduling solutions, $M(O_{i,j})$ the alternative number of machines of each operation $O_{i,j}$, $L$ is the total number of operations of all jobs and $Dist$ is the dissimilarity distance. The distance is calculated firstly by measuring the difference between the machine assignment vectors $MA_1$ and $MA_2$ which is in order of $O(n)$. So, if there is a difference, $Dist$ is incremented by $M(O_{i,j})$ (is the number of possible $n$ placement for each operation on its machine set, which is the alternative machine number of each operation $O_{i,j}$) because it is in the order of $O(n)$. Then, secondly, by verifying the execution order difference of the operation sequence vectors $OS_1$ and $OS_2$ which is in order of $O(1)$. So, if there is a difference, $Dist$ is incremented by 1 because it is in the order of $O(1)$. We give here how to proceed in Algorithm 1:

**Algorithm 1** How to calculate the dissimilarity distance between two solutions

1: procedure
2: $Dist \leftarrow 0, k \leftarrow 1$
3: for $k$ from 1 to $L$ do
4: if $Chrom1(MA_1(k)) \neq Chrom2(MA_2(k))$ then
5: $Dist \leftarrow Dist + M(O_{i,j})$
6: end if
7: if $Chrom1(OS_1(k)) \neq Chrom2(OS_2(k))$ then
8: $Dist \leftarrow Dist + 1$
9: end if
10: end for
11: return $Dist$
12: end procedure
3.1.3. Selection operator

The selection operator is used to select the best parent individuals to prepare them to the crossover step. This operator is based on a fitness parameter allowing to analyze the quality of each selected solution. But progressively the fitness values will be similar for the most individuals. That is why, we integrate the neighborhood parameter, where we propose a new combined parent selection operator named Fitness-Neighborhood Selection Operator (FNSO) allowing to add the dissimilarity distance criteria to the fitness parameter to select the best parents for the crossover step. The FNSO chooses in each iteration two parent individuals until engaging all the population to create the next generation. The first parent takes successively in each case a solution \( i \), where \( i \in \{1, \ldots, P\} \) and \( P \) is the total population size. The second parent obtains its solution \( j \) randomly by the roulette wheel selection method based on the two Fitness and Neighborhood parameters relative to the selected first parent, where \( j \in \{1, \ldots, P\} \setminus \{i\} \) in the \( P \) population and where \( j \neq i \). In fact, to use this random method, we should calculate the Fitness-Neighborhood total \( FN \) for the population, see Equation (5), the selection probability \( sp_k \) for each individual \( I_k \), see Equation (6), and the cumulative probability \( cp_k \), see Equation (7). After that, a random number \( r \) will be generated from the uniform range \([0,1]\). If \( r \leq cp_1 \) then the second parent takes the first individual \( I_1 \), else it gets the \( k^{th} \) individual \( I_k \in \{I_2, \ldots, I_P\} \setminus \{I_1\} \) and where \( cp_{k-1} < r \leq cp_k \). For Equations (5), (6) and (7), \( k = \{1,2,\ldots,P\} \setminus \{i\} \).

- The Fitness-Neighborhood total for the population:

\[
FN = \sum_{k=1}^{P} \left[ \frac{1}{(C_{max}[k] \times \text{Neighborhood}[i][k])} \right] 
\]  

(5)

- The selection probability \( sp_k \) for each individual \( I_k \):

\[
sp_k = \frac{1}{(C_{max}[k] \times \text{Neighborhood}[i][k])} \times \frac{FN}{FN} 
\]  

(6)

- The cumulative probability \( cp_k \) for each individual \( I_k \):

\[
cp_k = \sum_{h=1}^{k} sp_h 
\]  

(7)

3.1.4. Crossover operator

The crossover operator has an important role in the global process, allowing to combine in each case the chromosomes of two parents in order to obtain new individuals and to attain new better parts in the search space. In this work, this operator is applied with two different techniques successively for the parent’s chromosome vectors MA and OS. For the machine vector crossover, a uniform crossover is used to generate in each case a mixed vector between two machine vector parents, Parent1-MA1 and Parent2-MA2, allowing to obtain two new children, Child1-MA1’ and Child2-MA2’. This uniform crossover is based on two assignment cases, if the generated number is less than 0.5, the first child gets the current machine value of parent1 and the second child takes the current machine value of parent2. Else, the two children change their assignment direction, first child to parent2 and the second child to parent1. For the operation vector crossover, an improved precedence preserving order-based on crossover (iPOX), inspired from, is adapted for the parent operation vector OS. This iPOX operator is applied following four steps, a first step is selecting two parent operation vectors \((OS_1, OS_2)\) and generating randomly two job sub-sets \(JS_1/JS_2\) from all jobs. A second step is allowing to copy any element in \(OS_1/OS_2\) that belong to \(JS_1/JS_2\) into child individual \(OS’_1/OS’_2\) and retain them in the same position. Then the third step deletes the elements that are already in the sub-set \(JS_1/JS_2\) from
assigning each individual Indiv parameter which is the dissimilarity distance between individuals. The clustering operator starts by selecting one individual in each of them. For each case, we fixe an individual Indiv(i) and we verify successively for each next individual Indiv(j) from the remaining population (where \( i \neq j \)) if the dissimilarity distance \( Dist \) between Indiv(i) and Indiv(j) is less than or equal to a fixed threshold \( Dist_{fix} \) (representing a percentage of difference \( X\% \) relatively to \( Dist_{max} \), see Equation (8)) and where \( Cluster(Indiv(i)) \neq Cluster(Indiv(j)) \). If it is the case, \( Merge(Cluster(Indiv(i)), Cluster(Indiv(j))) \), else continue the search for new combination with the remaining individuals. The stopping condition is by browsing all the population individuals, where we obtained at the end \( N \) Clusters.

\[
Dist_{fix} = Dist_{max} \times X\%
\]  

3.1.5. Mutation operator

The mutation operator is integrated to promote the children generation diversity. In fact, this operator is applied on the chromosome of the new children generated by the crossover operation. Also, each part of a child chromosome \( MA \) and \( OS \) has separately its own mutation technique. The machine mutation operator uses a random selection of an index from the machine vector \( MA \). Then, it replaces the machine number in the selected index by another belonging to the same alternative machine set. The operation mutation operator selects randomly two indexes \( index1 \) and \( index2 \) from the operation vector \( OS \). Next, it changes the position of the job number in the \( index1 \) to the second \( index2 \) and inversely.

3.1.6. Replacement operator

The replacement operator has an important role to prepare the remaining surviving population to be considered for the next iterations. This operator replaces in each case a parent by one of its children which has the best fitness in its current family.

3.1.7. Clustering operator

By finishing the maximum iteration number \( MaxIter \) of the genetic algorithm, the Scheduler Agent applies a clustering operator using the hierarchical clustering algorithm of 21 to divide the final population into \( N \) Clusters to be treated by the Cluster Agents in the second step of the global process. The clustering operator is based on the neighbourhood parameter which is the dissimilarity distance between individuals. The clustering operator starts by assigning each individual Indiv(i) to a cluster \( CL_i \), so if we have \( P \) individuals, we have now \( P \) clusters containing just one individual in each of them. For each case, we fix an individual Indiv(i) and we verify successively for each next individual Indiv(j) from the remaining population (where \( i \neq j \)) if the dissimilarity distance \( Dist \) between Indiv(i) and Indiv(j) is less than or equal to a fixed threshold \( Dist_{fix} \) (representing a percentage of difference \( X\% \) relatively to \( Dist_{max} \), see Equation (8)) and where \( Cluster(Indiv(i)) \neq Cluster(Indiv(j)) \). If it is the case, \( Merge(Cluster(Indiv(i)), Cluster(Indiv(j))) \), else continue the search for new combination with the remaining individuals. The stopping condition is by browsing all the population individuals, where we obtained at the end \( N \) Clusters.

\[
Dist_{fix} = Dist_{max} \times X\%
\]  

3.2. Cluster Agents

Each Cluster Agent \( CA_i \) is responsible to apply successively to each cluster \( CL_i \) a local search technique which is the Tabu Search algorithm to guide the research in promising regions of the search space and to improve the quality of the final population of the genetic algorithm. In fact, this local search is executed simultaneously by the set of the CAs agents, where each CA starts the research from an elite solution of its cluster searching to attain new more dominant individual solutions separately in its assigned cluster \( CL_i \). The used Tabu Search algorithm is based on an intensification technique allowing to start the research from an elite solution in a cluster \( CL_i \) (a promising part in the search space) in order to collect new scheduling sequence minimizing the makespan. Let \( E \) the elite solution of a cluster \( CL_i \), \( E^* \in N(E) \) is a neighbor of the elite solution \( E \), \( GL_i \) is the Global List of each \( CA_i \) to receive new found elite solutions by the remaining CAs, each \( CL_i \) plays the role of the tabu list with a dynamic length and \( Cmax \) is the makespan of the obtained solution. So, the search process of this local search starts from an elite solution \( E \) using the move and insert method of 5, where each Cluster Agent \( CA_i \) changes the position of an operation \( O_{i,j} \) from a machine \( m \) to another machine \( n \) belonging to the same alternative machine set of this selected operation \( O_{i,j} \), searching to generate new scheduling combination \( E^* \in N(E) \). After that, verifying if the makespan value of this new generated solution \( Cmax(E^*) \) dominates \( Cmax(E) \) \( (Cmax(E^*) < Cmax(E)) \), and if it is the case \( CA_i \) saves \( E^* \) in its tabu list (which is \( CL_i \)) and sends it to all the other CAs agents to be placed in their Global Lists \( GLs(E^*, CA_i) \), to ensure that it will not be used again by them as a search point. Else continues the neighborhood search from the
current solution $E$. The stopping condition is by attaining the maximum allowed number of neighbors for a solution $E$ without improvement. We give here how to proceed in Algorithm 2:

**Algorithm 2** The local search process

1: procedure
2: $E \leftarrow \text{Elite}(CL_i)$
3: while $N(E) \neq 0$ do
4: $E \leftarrow \text{Move-and-insert}(E) \mid E' \in N(E) \mid E' \notin GL_i$
5: if $C_{\text{max}}(E') < C_{\text{max}}(E)$ and $E' \notin GL_i$ then
6: $E \leftarrow E'$
7: $GL_i \leftarrow E'$
8: Send-to-all($E'$, $CA_i$)
9: end if
10: end while
11: return $E$
12: end procedure

By finishing this local search step, the CAs agents terminate the process by sending their last best solutions to the SA agent, which considers the best one of them the global solution for the FJSP.

### 4. Experimental results

#### 4.1. Experimental setup

The proposed GATS+HM is implemented in java language on a 2.10 GHz Intel Core 2 Duo processor and 3 Gb of RAM memory, where we use the Integrated Development Environment (IDE) eclipse to code the algorithm and the multiagent platform Jade\textsuperscript{22} to create the different agents of our holonic model. To evaluate its efficiency, numerical tests are made based on two sets of well known benchmark instances in the literature of the FJSP: the Brandimarte data\textsuperscript{3} consisting of 10 problems considering a number of jobs ranging from 10 to 20, which will be processed on a number of machines ranging from 4 to 15. The Hurink edata\textsuperscript{4} consisting of 40 problems (la01-la40) inspired from the classical job shop instances of\textsuperscript{23}, where three test problems are generated: rdata, vdata and edata which is used in this paper.

Due to the non-deterministic nature of the proposed algorithm, we run it five independent times for each one of the two instances\textsuperscript{3} and\textsuperscript{4} in order to obtain significant results. The computational results are presented by five metrics such as the best makespan ($\text{Best}$), the average of makespan ($\text{Avg \ Cmax}$), the average of CPU time in seconds ($\text{Avg \ CPU}$), and the standard deviation of makespan ($\text{Dev \ %}$) which is calculated by Equation (9). The $Mko$ is the makespan obtained by Our algorithm and $Mkc$ is the makespan of an algorithm that we chose to Compare to.

$$\text{Dev} = \frac{(Mkc - Mko)/Mkc}{\times 100\%}$$ (9)

The used parameter settings for our approach are adjusted experimentally and presented as follow: the crossover probability = 1.0, the mutation probability = 1.0 and the maximum number of iterations = 1000. The population size ranged from 15 to 400 depending on the complexity of the problem.

#### 4.2. Experimental comparisons

To show the efficiency of our GATS+HM algorithm, we compare its obtained results from the two previously cited data sets with other well known algorithms in the literature of the FJSP. The chosen algorithms are the TS of\textsuperscript{3}, the N1-1000 of\textsuperscript{4} (with their literature lower bound LB), the MATSLO+ of\textsuperscript{12} and the MATSPSO of\textsuperscript{13}. The different comparative results are displayed in the Tables 2 and 3, where the first column takes the name of each instance, the second column gives the size each instance, with $n$ the number of jobs and $m$ the number of machines ($n \times m$), and the
remaining columns detail the experimental results of the different chosen approaches in terms of the best \( \text{Cmax}(\text{Best}) \) and the standard deviation \( (\text{Dev} \%) \). The bold values in the tables signify the best obtained results.

### 4.2.1. Analysis of the comparative results

By analyzing the Table 2, it can be seen that the GATS+HM obtains nine out of ten best results for the Brandimarte instances. In fact, our approach outperforms the TS in nine out of ten instances. Moreover, for the comparison with MATSLO+, our GATS+HM outperforms it in eight out of ten instances. Furthermore, the MATSPSO attained the best result for the MK01 instance, but our approach obtains a set of solutions better than it for the remaining instances.

By solving this first data set, our GATS+HM attains the same results obtained by some approaches such as the MK01 for MATSLO+, the MK02 for MATSPSO and the MK08 for all methods. From Table 3, the obtained results show that the GATS+HM obtains seven out of ten best results for the Hurink edata instances (la01-la05) and (la16-la20). Indeed, our approach outperforms the N1-1000 in eight out of ten instances. Moreover, our GATS+HM outperforms the MATSLO+ in seven out of ten instances. For the comparison with the literature lower bound LB, the GATS+HM attains the same results for the la01, la02, la04, la05, la16, la17 and la20 instances, but it gets slightly worse result for the la03, la18 and la19 instances. Furthermore, by solving this second data set, our GATS+HM attains the same results obtained by the chosen approaches such as in the la01 for the MATSLO+; in the la02 for the N1-1000 and the MATSLO+; in the la05 for the N1-1000 and the MATSLO+.

By analyzing the computational time in seconds and the comparison results of our algorithm in term of makespan, we can distinguish the efficiency of the new proposed GATS+HM relatively to the literature of the FJSP. This effi-
ciency is explained by the flexible selection of the promising parts of the search space by the clustering operator after the genetic algorithm process and by applying the intensification technique of the tabu search allowing to start from an elite solution to attain new more dominant solutions.

5. Conclusion

In this paper, we present a new metaheuristic hybridization within a holonic multiagent model, called GATS+HM, for the flexible job shop scheduling problem (FJSP). In this approach, a Neighborhood-based Genetic Algorithm is adapted by a Scheduler Agent (SA) for a global exploration of the search space. Then, a local search technique is applied by a set of Cluster Agents (CAs) to guide the research in promising regions of the search space and to improve the quality of the final population. To measure its performance, numerical tests are made using two well known data sets in the literature of the FJSP. The experimental results show that the proposed approach is efficient in comparison to others approaches. In the future work, we will search to treat other extensions of the FJSP, such as by integrating new transportation resources constraints in the shop process. So, we will make improvements to our approach to adapt it to this new transformation and study its effects on the makespan.

References

1. Sonmez, A.I., Baykasoglu, A.. A new dynamic programming formulation of (nm) flow shop sequencing problems with due dates. International Journal of Production Research 1998;36(8):2269–2283.
2. Garey, M.R., Johnson, D.S., Sethi, R.. The complexity of flow shop and job shop scheduling. Mathematics of Operations Research 1976; 1(2):117–129.
3. Brandimarte, P.. Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 1993;41(3):157–183.
4. Hurink, J., Jurisch, B., Thole, M.. Tabu search for the job-shop scheduling problem with multi-purpose machines. Operations Research Spektrum 1994;15(4):205–215.
5. Mastroiilli, M., Gambardella, L.. Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling 2000; 3(1):3–20.
6. Bozejko, W., Uchronski, M., Wodecki, M.. The new golf neighborhood for the flexible job shop problem. In: In Proceedings of the International Conference on Computational Science. 2010, p. 289–296.
7. Chen, H., Ilow, J., Lehmann, C.. A genetic algorithm for flexible job-shop scheduling. In: In Proceedings of the IEEE International Conference on Robotics and Automation. 1999, p. 1120–1125.
8. Gao, J., Sun, L., Gen, M.. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers and Operations Research 2008;35(9):2892–2907.
9. Zhang, C., Gu, P., Jiang, P.. Low-carbon scheduling and estimating for a flexible job shop based on carbon footprint and carbon efficiency of multi-job processing. Journal of Engineering Manufacture 2014;39(2):1–15.
10. Ziaee, M.. A heuristic algorithm for solving flexible job shop scheduling problem. The International Journal of Advanced Manufacturing Technology 2014;71(1-4):519–528.
11. Ennigrou, M., Ghédira, K.. Approche multi-agents basée sur la recherche tabou pour le job shop flexible. In: In 14ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle RFIA 2004, p. 28–30.
12. Ennigrou, M., Ghédira, K.. New local diversification techniques for the flexible job shop problem with a multi-agent approach. Autonomous Agents and Multi-Agent Systems 2008;17(2):270–287.
13. Henchiri, A., Ennigrou, M.. Particle swarm optimization combined with tabu search in a multi-agent model for flexible job shop problem. In: In Proceedings of the 4th International Conference on Swarm Intelligence, Advances in Swarm Intelligence. 2013, p. 385–394.
14. Glover, F., Kelly, J.P., Laguna, M.. Genetic algorithms and tabu search: Hybrids for optimization. Computers & Operations Research 1995;22(1):111–134.
15. Ferber, J.. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 1st ed.; 1999. ISBN 0201360489.
16. Botti, V., Giret, A.. ANEMONA: A Multi-agent Methodology for Holonic Manufacturing Systems. Springer Series in Advanced Manufacturing. Springer-Verlag; 2008. ISBN 978-1-84800-310-1.
17. Koesler, A.. The Ghost in the Machine. London, United Kingdom: Hutchinson; 1st ed.; 1967. ISBN 0090838807.
18. Calabrese, M.. Hierarchical-granularity holonic modelling. Doctoral thesis; Universita degli Studi di Milano; Milano, Italy; 2011.
19. Giret, A., Botti, V.. Holons and agents. Journal of Intelligent Manufacturing 2004;15(5):645–659.
20. Lee, K., Yamakawa, T., Lee, K.M.. A genetic algorithm for general machine scheduling problems. In: In Proceedings of the second IEEE international Conference on Knowledge-Based Intelligent Electronic Systems. 1998, p. 60–66.
21. Johnson, S.C.. Hierarchical clustering schemes. Psychometrika 1967;32(3):241–254.
22. Bellifemine, F., Poggi, A., Rimassa, G., Jade - a fipa-compliant agent framework. In: In Proceedings of the fourth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology. 1999, p. 97–108.
23. Lawrence, S.. Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Tech. Rep.; Graduate School of Industrial Administration, Carnegie-Mellon University; Pittsburgh, Pennsylvania; 1984.