Journal of Combinatorial Optimization (2020) 39:602–617
https://doi.org/10.1007/s10878-018-00373-y

A new leader guided optimization for the flexible job shop problem

Fraj Naifar[1,2] · Mariem Gzara[2,3,4] · Moalla Taicir Loukil[5]

Published online: 3 January 2019

Abstract
The FJSP is an extension of the classical job shop problem which has been proven to be among the hardest combinatorial optimization problems, by allowing an operation to be operated on more than one machine from a machine set, with possibility of variable performances. In this work, we have designed a co-evolutionary algorithm that applies adaptively multiple crossover and mutation operators. In the evolution process, all new generated individuals are improved by local search. Combined with a new leader tree guided optimization search, the hybrid algorithm has discovered 2 new optimal solutions for instances of Hurink et al. (Oper Res Spektrum 15(4):205–215, 1994). In general, the outcomes of simulation results and comparisons demonstrate comparable results. The leader guided optimization has shown its effectiveness for minimizing the makespan in a FJSP, but it is not limited to this environment.

Keywords Scheduling · Flexible job shop problem · Evolutionary algorithm · Leader guided optimization · Multiple crossovers

✉ Mariem Gzara
mariem.gzara@gmail.com

Fraj Naifar
fraj.naifar@gmail.com

Moalla Taicir Loukil
Tmoalla@ut.edu.sa

1 Digital Research Center of Sfax, Sfax, Tunisia

2 Multimedia InfoRmation Systems and Advanced Computing Laboratory MIRACL, Sfax, Tunisia

3 University of Monastir and Computer Science, Monastir, Tunisia

4 High School of Mathematics and Computer Science, Monastir, Tunisia

5 Faculty of Business Administration, Tabuk University, Tabuk, Kingdom of Saudi Arabia

1 Introduction

This paper deals with the flexible job shop problem (FJSP) which is a generalisation of the job shop problem (JSP) by assuming that an operation may be processed by more than one type of machines, with possibility of variable performances inside the set of machines.

In the FJSP, there are n independent jobs $J = \{1, \ldots, j, \ldots, n\}$ and a set of K machines $M = \{m_k, 1 \le k \le m\}$ available at time t_0. Each job j is composed of a predefined sequence of n_j operations O_{ij}, where O_{ij} denotes the ith operation of the job j. To each operation O_{ij} is associated a pool of machines m_{ij}. Given a machine m_k from m_{ij} than the execution time of O_{ij} on m_k is e_{ijk}. A scheduling is a definition for each operation O_{ij} of a machine m_k from its pool, a starting date s_{ij} and a completion time c_{ij} of O_{ij} on that machine. The most considered objective is the minimization of the maximum completion time (makespan). The FJSP has a strongly NP-hard nature since it includes two sub-problems: operation's assignment problem and classical job shop scheduling problem.

A feasible schedule of the FJSP can be modelled by a disjunctive graph where the set of nodes is the set of operations to which are added dummy starting and terminating nodes. Two types of arcs join the nodes. The conjunctive arcs connect two successive operations according to the job execution order. The disjunctive arcs connect two adjacent operations processed on the same machine. The makespan is the length of the longest path in the disjunctive graph. This path is said to be the critical path and a graph may admits more than one critical path. Critical operations are those belonging to a critical path.

Much research has addressed the FJSP and most are based on metaheuristics and heuristics. The efficiency of population based metaheuristic is always enhanced with local search algorithms (Gao et al. 2008; Gen et al. 2009; Zhang et al. 2011; Singh and Mahapatra 2016). Several neighbourhood structures, mainly based on the disjunctive graph model, were designed for the FJSP. Particularly, those based on the critical path have demonstrated their performance to converge to good optima (Hurink et al. 1994; Mastrolilli and Gambardella 2000). Given the literature of the FJSP, none approach has established its superiority among the others and the work of Mastrolilli and Gambardella (2000) still realises among the best results.

Actually, much research literature addresses evolutionary algorithms (EAs) to solve the FJSP because of their ability to perform global search and provide good solutions in a short computation time. However, evolutionary algorithms suffer from stagnation of the search after several genetic iterations due to the domination of few even one high efficient individual. To deal with, we have developed a new leader guided optimization (LGO) search technique. The main idea is to run independently many evolutionary instances where each one is guided by a leader. A leader is a powerful solution discovered in the previous evolution process. We have combined the LGO with a new designed co-evolutionary algorithm (CEA) to solve the FJP. The CEA applies adaptively multiple crossover and mutation operators so as to ensure the effect of multiple evolution schemes into the same algorithm. The hybrid algorithm has shown its performance in the resolution of the FJSP comparing to the literature results. Two new optimal solutions are found in rdata instances of Hurink et al. (1994).

The remainder of the paper is organized as follows: Sect. 2 presents the leader guided optimization. Section 3 describes the co-evolutionary algorithm. Section 4 presents the hybridization scheme and the results.

2 Leader guided optimization

The main idea is to guide the search process by a leader. The leader has a good performance or it contains some promising characteristics. The leader is declared as a winner of a race. In each race, a group of individuals are guided by the leader. These individuals evolve during a race. They can whether exchange experience or adopt evolution and diversification strategies. The winner of a race is saved as a leader. Later, it will initiate another race with a new generated group of individuals. One individual has a limited extreme performance period during his life. Thus, he has a limited number of races to run. The winners of the races are stored in the memory. They form a bank of leaders.

When a leader reaches its performance limit, which is a number of races that it performs during its life, a new leader is selected in the bank of leaders to become the new race leader. The race's leader enters the race with the sole purpose of guiding the other individuals and not to win the race. During the race, the individuals evolve by combination and diversification operators through a population based search technique. A race may be an independent run of population based search technique guided by the leader.

The leader guided optimization uses the following mechanisms:

- Initiation of the bank of leaders. These leaders may be generated by different ways. They can be randomly generated than locally optimized or obtained by using some rapid heuristics.
- Leader selection: this mechanism selects from the bank of leader the one that will guide the next races. This individual may be the most performing or selected according to a priority rule such as LIFO or FIFO.
- Population initialization: the initial population of the race is composed of a set of new generated solutions added to the race's leader. These solutions are created independently of the leader whether randomly or by constructive heuristics.
- A race: a race is a population based metaheuristic which allow the combination of the leader's features with other individuals.

The global functioning of the leader guided optimization is described in the Algorithm 1.

Algorithm 1: leader guided optimization

```
Step 1: bank_leaders := Initiate the bank of leaders
Step 2: E_current := select an unlocked leader from the bank
Step 3: while E_current is a race leader do
            E_new :=perform a race guided by E_current and returns the race's winner.
            if E_new is best then  E_current
                Then insert E_new in the bank of leaders
        End while
        E_current is locked
Step 4: if the stopping condition is reached returns the best leader, otherwise go to step 2.
```

The leaders are stored in a k array tree data structure where k is a parameter of the algorithm. The root of the tree is a null node. Each node of the tree has at most k children. For a given node of the tree, its children are leaders that have won a race guided by their parent node. In fact, a leader performs at most k races in its life. Only the winners of the races are stored as leaders. It is obvious that the child's efficiency is higher than that of its parent. The children of a node are stored from right to left by decreasing order of performances. The children of the root node are generated independently by an initiation mechanism of the bank of leaders. The leaders that have accomplished their races are locked and the others are unlocked. The new leader is selected from the unlocked ones. This data structure injures depth and width search. One can whether starts by a node and performs the races on his children first or on the next node in the same level of the tree.

Let suppose we have a minimization problem and we have generated three initial leaders (x_1, 100), (x_2, 107) and (x_3, 108). The winners of three independent races guided by $x_1$are: (x_{11}, 94), (x_{12}, 90) and (x_{13}, 93). Only two races guided by x_2 have discovered new leaders: (x_{21}, 94) and (x_{22}, 96). The races guided by x_3 have generated the leader (x_{31}, 85). The obtained tree is given in Fig. 1.

We have applied the leader guided search to mathematical optimization functions. These functions are used to evaluate optimization algorithms.

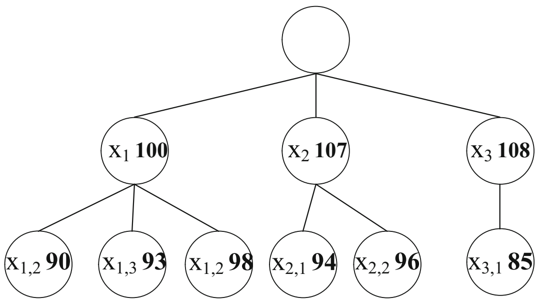

Fig. 1 The bank of leader

We have considered the following known minimization functions:

Rosenbrock function:

$$\sum_{i=1}^{d-1}\left(1-x_i^2\right)^2+100\left(x_{i+1}-x_i^2\right)^2,\quad -2.048\le x_i\le 2.048$$

The global optimum is $f_{min}=0$ for the point (1, 1) in 2D dimension. The parameter d fixes the number of dimensions.

Eggcrate function:

$$g(x,y)=x^2+y^2+25\left(\sin^2 x+\sin^2 y\right),\ (x,y)\in[-2\pi,2\pi]\times[-2\pi,2\pi]$$

The eggcrate function is multimodal. The global minimum is $g_{min}=0$ at point (0, 0).

Michalewicz function:

$$f(x)=-\sum_{i=1}^{d}\sin(x_i)\left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m},\ (m=10)$$

This function admits d! local optima in the interval $0\le xi\le \Pi$ for i = 1, …, d. The global minimum in 2D dimension is $f_{min}=-1.801$.

We have considered a standard genetic algorithm with binary encoding, one crossover and one point mutation and wheel selection. A race in the leader-guided optimization is a genetic algorithm of 100 individuals with the probabilities respectively of crossover 0.65 and mutations 0.1.

We run 18,000 generations of the standard genetic algorithm. For the LGO we run 6 races of 3000 generations of the GA. Every leader runs two races in its life. The results are given in the following table.

For the Rosenbrock function the bank of leader is initiated with the individuals (s_1; 11.2) and (s_2; 1.28). s_2 is better than s_1. Thus s_2 guides two races that are winned by (s_{21}; 10.07) and (s_{22}; 1.17). The solution s_{22} is the best child. So, it guides the following two races that were won by the solutions (s_{221}; 0.6) and (s_{222}; 7.1).

By comparison to the standard genetic algorithm, the LGO has performed better for the 3 problems (Table 1). For the eggarte function, the LGA reaches an optimum equal to 0.03 (global optimum = 0) while the GA returns the value 4.29. For the Michalewicz function, the optimal value is − 1.801. The LGO (− 1.71) is nearest to the optimum than the GA (− 1.61). For the Rosenbrock, both algorithms returns the same value. The LGO has proved its efficiency when applied to well known mathematical optimization functions. In the next section, we demonstrate its efficiency in the resolution of a combinatorial optimization problem, which is the flexible job shop problem.

3 A co-evolutionary algorithm for the FJSP

The adaptation of the evolutionary algorithm to the FJSP is mentioned as co-evolutionary because multiple crossover and mutation operators are used to obtain

Table 1 Results for the test functions

Function	LGO	GA
Rosenbrockk	0.6	0.6
Eggcrate	0.03	4.29
Michalewicz	− 1.71	− 1.61

a diversified set of combination and diversification schemes. These genetic operators are applied with adapted frequency that depends on their performance to generate new efficient solutions. Each new generated solution is optimized locally by local search.

The overall structure of the co-evolutionary process can be described by the Algorithm 2.

Algorithm 2: Co-evolutionary algorithm

```
(1)   Set parameters
(2)   Initial population generation
          (2.1) Generate $n_1$ solutions by deterministic combined rules
          (2;2) Generate $n_2$ solutions by stochastic rules
(3)   Evaluate makespan
(4)   Selection of parents by tournament selection of size $\lambda$
(5)   Opc:=Crossover operator selection
(6)   Combine parents by opc
(7)   Complete children by scheduling heuristic
(8)   Update crossovers probabilities
(9)   Opm:=Mutation operator selection
(10)  Mutate children by opm
(11)  Apply local search
(12)  Update mutation probabilities
(13)  Return to 4 until formation of all desired descendents
(14)  Update best solution
(15)  Select survivals from parents and children
(16)  Update $\lambda(t)$
(17)  Stop if termination condition else go to 4
```

Coding the coding is similar to the parallel machine encoding proposed in Mesghouni et al. (2004). A solution is coded as a table of K lists. The kth list is the sequence of the machine m_k. This encoding is direct since it gives directly the assignment and the sequencing of the operations. To obtain the final scheduling, each operation receives the earliest possible starting time.

Scheduling heuristics the solutions are constructed through the application of scheduling and assignment rules. Two priority rules are used for operation selection:

- Select randomly one operation from the queue list.
- Select the operation that has the least available time.

The machine selection is based on three assignment rules:

- Assign the operation to a random machine from its pool
- Assign the operation to the machine that can start it earlier
- Assign the operation to the least loaded machine from its pool

Six combined heuristics are then used to generate the initial solutions where two of them are deterministic and the others are stochastic. The obtained solutions are improved by local search before their insertion in the initial population.

Selection the tournament selection is applied with variable size of the tournament. The selection incorporates the elite model since the best solution is always transferred to the next generation. A high pressure of selection is authorized in the start of the search to accelerate the convergence then binary tournament selection is performed to avoid search stagnation. The pressure of selection is decreased by one every T generations ($\lambda(t) = \max(\lambda(t-1) - 1, 1)$ if t modulo $T = 0$), where T is a parameter of the CEA.

Local search local search is performed on every new generated solution to improve locally the solution generation process. The genetic evolution gives promising starting points to the local search to intensify the search in many directions.

The local search is based on the disjunctive graph model of Roy and Sussmann (1964) and the neighbourhood of Nowicki and Smutnicki (1996). It starts by decomposing the critical path into maximal blocks of adjacent critical operations being treated by the same machine. A neighbouring solution is obtained by a swap of the last two and first two block operations.

Stop criterion the algorithm is stopped if a maximum number of iterations is reached or the best solution is not changed for a fixed number of iterations.

Multiple crossover evolution the combination of the chromosomes is performed via distinct crossover operators. The crossovers exchange sequences between parents to form new encodings in different ways. The generated encodings are feasible but incomplete. To preserve feasibility, operations that violate the scheduling constraints, are inserted into a waiting list to be re-inserted later by the scheduling heuristic. The generation of partial encoding by combination of the parent's encoding avoids the use of the reparation techniques which may modify the assignment or the operation's sequencing of their parents. Some good features may be omitted by reparation and lost. Four crossover operators are proposed.

One Point Crossover with Priority List (OPCPL) Let P_1 and P_2 be a couple of mates and k be a cut point on the set of machines ($1 < k < K$). The cut point divides the mate P_1 into two blocks of machine sequences A and B (respectively P_2 into C and D). The child E_1 inherits from the parent P_1 the block A and from the parent P_2 (respectively the parent P_1), a part D' of the block D. Operations in D that violates the scheduling constraints if they are copied to the encoding of the child E_1 are inserted into a waiting list. In addition, omitted operations are added to the waiting list. If the waiting list L is not empty ($L \neq \Phi$), the formation of the complete encoding and the computation of the scheduling are performed by the scheduling heuristic. The child E_2 is formed in the same way as E_1 while inverting the role of P_1 and P_2 in the combination process.

Two Points Crossover with Priority List (TPCPL) Let P_1 and P_2 be a couple of mates and two cut points k_1 and k_2 ($1 < k_1 < k_2 < K$) to delimitate the crossover zone. The parent is cut into three blocks of machines. The block A of the machines m_i $i \in <1, k_1>$, block B of machines m_i $k_1 < i \leq k_2$, and the block C of machines m_i, $k_2 < i \leq K$. By the same way, the parent P_2 is divided into three blocks D, E and F. The child E_1 receives from P_1 the same sequences on the machines inside the crossover zone and from P_2, those outside the crossover zone without violating the scheduling constraints. The obtained child is an incomplete encoding finalized by the scheduling heuristic. In fact, only feasible operations are copied from parent P_2 to child E_1. By the same way, the encoding of the child E_2 is composed of the block E from P_2 and feasible parts of blocks B and C from P_1.

Selective Machine Sequences Crossover (SMSC) Only one offspring is produced by the SMSC operator. This child inherits, for each machine, the sequence satisfying one of the following criteria: the minimum completion time, the least loaded machine or one randomly chosen machine amongst the two parents. The obtained encoding is partial since there are omitted and not feasible operations which are stored in a waiting list and after that rescheduled by the scheduling heuristics.

Multi-ParentCrossover (MPC) To guarantee a high level of diversification the MPC try to join sequences from three parents to construct a child. Once, a crossover zone is created by randomly generating two positions k_1 and k_2 ($1 \leq k_1 < k_2 < K$), the child E_1 receives from P_1 the same sequences inside the crossover zone, from P_2 the k_1 first sequences and from P_3 the last sequences. This step is achieved by setting up a list L of un-rooted operations. The MPC takes end when all pending operations are inserted into their appropriate machines by the scheduling heuristics.

Multiple mutation evolution The proposed mutation operators are trivial heuristics that can guide the search to new promising directions. We have designed multiple mutation operators to conjugate their effect on the evolutionary process. All the mutation operators are designed for multipurpose operations except the swap mutation on one machine and the rescheduling of the last finishing job, they don't modify the operation's assignments to the machines. Six mutation operators are designed.

Mutation of operation reschedule a multi-purpose operation on another machine from its pool.

Mutation of the loaded machine reschedule a multipurpose operation from the sequence of the most loaded machine on another machine from its pool.

Mutation of the last finishing job if the most loaded machine executes a multipurpose operation of the last finishing job then reschedule it if possible.

Mutation by job rescheduling extract all the operations of the last finishing job and reschedule without assignment modification.

Swap mutation on one machine swap two operations on a machine sequence.

Swap mutation on two machines swap two multipurpose operations each one from a machine.

All the mutation operators proceed by removing one or many operations from a given scheduling than by rescheduling the removed operations. A candidate operation is inserted in the first available position in the sequence of the machine. Let O_{ij} be an operation to be inserted in the sequence of the machine m_k. We start by seeking the last job predecessor of O_{ij} executed by the machine m_k. Let $O_{i'j}$ be that job predecessor if it exists. O_{ij} can't be inserted before $O_{i'j}$.

By the same way, the operation O_{ij} can't be processed after one of its job successor on the machine m_k. Let $O_{i''j}$ be the first job successor of O_{ij} on the machine m_k.

We look know for the operations comprised between $O_{i'j}$ and $O_{i''j}$. Let O be one of these operations. If a job predecessor of O is placed after a job successor of O_{ij} in the sequence of another machine $m_{k'}$, the insertion of O_{ij} after O generates a cycle and this insertion is forbidden. The operation O_{ij} can be inserted before the first operation outside its job that hasn't one of its job predecessors executed after a job successor of job O_{ij}. Let O' be that operation.

The operation O_{ij} is inserted in the first possible feasible position: after both $O_{i'j}$ and O' and before $O_{i''j}$.

Adaptive operator selection a dynamic probability of application is assigned to each evolutionary operator (mutation, crossover). The operator that performs well during the previous iterations will have more chance to be applied. When an operator generates an offspring better than its parents, the probability of that operator is increased. During the search weaker operators are penalized and stronger ones are encouraged for a given problem instance. The probability of application of a given operator is reinforced if the later performs well otherwise it will be decreased. The operators aren't allowed to die and neither operator directs alone the search. The probabilities are maintained in a pre-fixed interval. To do so we have adapted the principle of the Adaptive Pursuit Algorithm (APA) as described in Thierens (2007).

Given A operators $i = 1 \ldots A$. We associate to the operator i, a count t_i that equals the number of times the operator i was applied. The operator i has also a probability of application P_i such that $(0 \le P_i(t) \le 1; \sum_{i=1}^{A} P_i(t) = 1$. Every time an operator i is applied, its quality estimator is updated as follows:

$$Q_i(t_i + 1) = Q_i(t_i) + \alpha\left[R_i(t_i) - Q_i(t_i)\right] \tag{1}$$

where $\begin{cases} R_i[t_i] := R_{Fix} + \left(\frac{C_p}{C_c}\right) - 1, & \text{if the operator } i \text{ improves the solution quality} \\ R_i[t_i] := 0, & \text{otherwise} \end{cases}$

where the quantity $R_i(t_i)$ is a reward received by the operator i when its application improves the results. This reward depends on the objective function. The terms C_p and C_c are respectively the makespan of the parent and the descendent after the application of the operator i. In case of a mutation operator, the child is the mutated individual. The parameter α is an adaptation rate: $0 < \alpha \le 1$.

Initially, all the operators have equal probabilities to be applied ($P_i(0) = 1/A, \forall i \in [1, A]$).

When a given operator has been applied a number of times multiple of max Q, the probability vector is updated.

$$i* = \arg \max_{i \in 1 \ldots A} (Q_i(t_i + 1)) \tag{2}$$

$$P_{i*}[t_{i*} + 1] := P_{i*}[t_{i*}] + \beta * (P_{max} - P_{i*}[t_{i*}] \tag{3}$$

$$\text{For} \quad i \neq i*, \; P_i[t_i + 1] := P_i[t_i] + \beta * \left(P_{min} - P_i[t_i]\right) \tag{4}$$

The parameters of the adaptive pursuit are fixed after fine-tuning with: $\alpha = 0.85$, $\beta = 0.85$, $R_{Fix} = 0.9$. The parameters P_{max} is set as follow: $P_{max} = 1 - (A - 1) * P_{min}$ with $0 < P_{min} < 1$. This setting ensures that the best operator is applied with a probability near to 0.5 and that the others are equally probability applied. So, neither operator is neglected or dominates the evolution process.

4 Experimental results and discussion

The proposed method was implemented in C++ language on an i5 processor running at 2.4 GHz. The parameters of the EA were fixed after carrying out multiple fine tuning as follow: crossover probability = 0.71, mutation probability = 0.6, population size = 50, 100, 200 and 300 and generation number = 10,000). We have varied the mutation and crossover operators from 0 to 1 by step of 0.1. For each, combination we run the algorithm 5 times for a population of 50 individuals. The algorithm performs well for high rate of mutation and crossover. But it converges rapidly since each generated solution is improved by local search. A rate of reproduction comprised between 0.6 and 0.8 realizes a good compromise between exploration and exploitation in the evolutionary process.

Three FJSP different sets of instances are used: the instances “mt06”, “mt10”, and “mt20” are taken from Fisher and Thompson (1963). The three data sets of 40 instances “la01” to “la40” which are “edata” with the least amount of flexibility, “rdata” where the average size of m_{ij} is equal to 2 and “vdata” where the average reach m/2 (Hurink et al. 1994). We compare five algorithms: the evolutionary algorithm (EA), the co-evolutionary algorithm (CEA), the hybridization of the leader guided optimization with the CEA (LGO_CEA) and the tabu search TS of Mastrolilli and Gambardella (2000). The LGO_CEAruns multiple races guided by dominant solutions discovered in the previous search. Each race is a population of individuals that evolve according to the CEA.

To illustrate the behavior of the leader guided optimization when applied to the flexible job shop problem we give in Fig. 2 a branch of the leader tree for the problem la38 vdata from Hurink et al. (1994).

The initial leader has a makespan of 997. He has leaded 7 independent races. Its most left child is the best one among its 7 children. This later leads also 7 races but he was dominated only in 3 of them. Thus, the node x_1 has only three children where the best of them is x_{11}. This individual will guide the next races. At this step of the

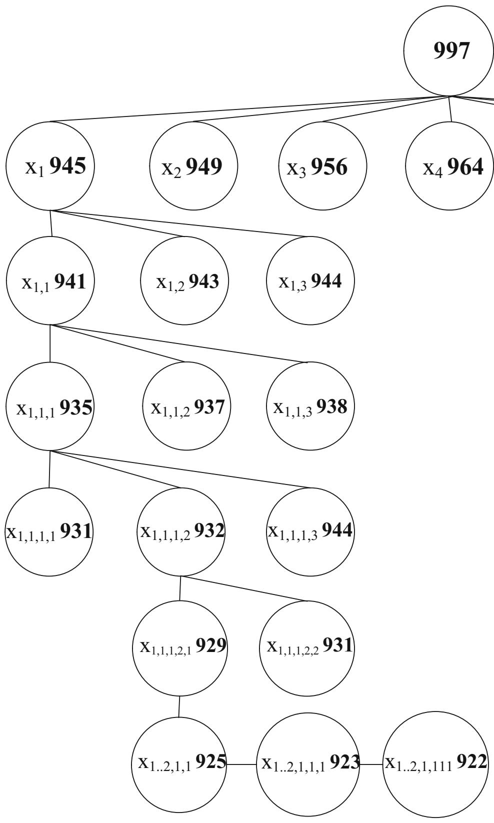

Fig. 2 Leader tree for the instance la38 vdata

Table 2 Comparison of the MRE of algorithms EA, CEA, LGO_CEA, GA and TS for solving FJS

MRE	EA	CEA	LGO_CEA	GA	TS
HurinkEdata	6.84	4.09	2.82	6	1.99
HurinkRdata	4.28	2.9	1.87	4.42	1.16
HurinkVdata	1.1	0.64	0.21	2.04	0.1

algorithm, the bank of leaders contains 11 leaders where 2 of them are locked since they have still compete their races.

In Table 2, we compute the mean of relative error to the optimum for the 4 algorithms and for each test instances set. In Table 3, we give the number of found solutions for a given deviation from the optimum and we give the number of optima found.

The algorithm EA is a multiple crossover and mutation genetic algorithm. The EA selects randomly the reproduction operator to apply for crossover or mutation. It uses the same operators as those implemented in the CEA. The algorithm CEA implements the guided selection strategy that favours the application of the best performant operators during the search.

Table 3 Performance comparison between EA, CEA, LGO_CEA and TS sets for solving FJSPs

Problem	Algorihm	Nb Optima	$0 < \text{Val} < 1$	$1 \leq \text{Val} < 3$	$3 \leq \text{Val} < 10$	$10 \leq \text{Val} < 17$	$17 \leq \text{Val} < 21$
Hurink Edata	EA	8	2	3	19	8	3
	CEA	9	7	6	16	4	1
	LGO_CEA	20	5	3	11	4	0
	TS	26	2	4	7	4	0
Hurink Rdata	EA	5	14	7	6	11	0
	CEA	9	14	3	14	0	0
	LGO_CEA	17	9	7	10	0	0
	TS	16	15	4	8	8	0
Hurink Vdata	EA	9	22	5	7	0	0
	CEA	13	21	7	2	0	0
	LGO_CEA	28	11	4	0	0	0
	TS	32	11	0	0	0	0

The CEA discovers more optima than the EA for the three test instances. The mean deviation to lower bounds is also minimised. Among 129 instances, the CEA outperforms the EA on 78 instances, realizes the same performance on 47 and it is less performant only on 3 instances. This result confirms the efficiency of the adaptive selection strategy of the genetic operators.

The CEA performs better when the flexibility increases. This result is expected. The heuristics and the operators that consider the assignment sub-problem will have no effect and a lack of diversity will slow down the convergence. The CEA realised near results to those of TS for the vdata instances. The mean error deviation is comprised between 0 and 1 for 18 instances among 43. The CEA is near the optimal performances on a large set of instances.

We combine the CEA with the LGO to escape local optima by re-starting the search while guiding the population by a dominant individual discovered in previous searches. The hybrid algorithm LGO_CEA outperforms the CEA on all the test instances. LGO_CEA succeeds to characterise 65 optimal solutions where 31 among them were found by the CEA. For the remaining instances, LGO_CEA is nearest to the optimal than the CEA. Tables 2 and 3 show that the new Leader Guided Optimisation when combined with the CEA has brought remarkable ameliorations in the obtained results due to the enhanced exploration of the search space. The leader tree guided optimization has allowed a deep oriented exploration of the search space and thus has discovered always more efficient solution then the CEA if applied alone.

As for the CEA, the LGO_CEA realizes better performance when the flexibility increases. The number of optimal solutions found is higher for the test instances vdata by comparison to edata and rdata. The mean relative error decreases when the flexibility increases, it is lower for the vdata test instances. The scheduling heuristics and the genetic operators that consider both the scheduling and the assignment problem have a lower diversification effect as the flexibility decreases.

By comparison to the TS results, the LGO_CEA discovers two new optimal solutions for the problem LA01 and LA15 (Fig. 3) in rdata set. We can also notice that in term of number of optimal solutions, LGO_CEA is very close to TS of Mastrolilli and Gambardella (2000) in Edata and Vdata instances but exceeds it in Rdata. Additionally, our proposed method outperforms GA of Pezzella et al. (2008) for all types of instances. It is interesting to note that our approach finds two new optimal solutions: La01 and La15 in Hurinkrdata as mentioned.

In Table 4, we give the five number summary of the makespan and the relative standard deviation. This statistics measures are obtained from 20 independent runs of the LGO_CEA on four test instances la25 from edata and la22, la28 and la30 from vdata.

For the problem la25 the standard deviation is 2.03% of the mean so the obtained performances are tightly clustered around the mean. For the problems la28 and la30, the RSD is less than 1% so the values are around the mean. For the la22, the RSD is also low. The LGO_CEA realizes near performances when launched independently many times on the same test instances. We conclude that the algorithm is robust.

The table below shows the execution time of LGO_CEA and CEA. The LGO_CEA runs races guided by the leaders until the bank of leaders becomes empty. Each race is an independent execution of the CEA where a leader from the bank joins the initial

```
The optimal solution for the problem LA01 in rdata problem
machine 1: 1 5 0 83 83**2 1 83 136 53**2 8 136 196 60**2 4 196 251 55**4 6 251 343 92**
4 2 343 369 26**3 9 369 394 25**4 9 394 438 44**5 3 438 450 12**4 8 450 474 24**5 10 474 570 96**
machine 2: 1 6 0 54 54**1 1 54 75 21**2 3 75 173 98**3 3 173 215 42**2 9 215 264 49**
3 7 264 351 87**3 8 351 392 41**4 10 392 467 75**4 5 467 486 19**5 8 486 569 83**
machine 3: 1 2 0 21 21**1 8 21 59 38**2 6 59 102 43**2 7 102 179 77**2 5 179 213 34**
3 5 213 277 64**4 3 277 308 31**3 10 308 351 43**4 7 351 438 87**5 1 438 472 34**
5 9 472 570 98**
machine 4: 1 9 0 17 17**1 7 17 86 69**1 4 86 163 77**2 2 163 215 52**2 10 215 294 79**3 2 294 310 16**4
1 310 365 55**5 6 365 427 62**4 4 427 493 66**5 4 493 570 77**
machine 5: 1 3 0 39 39**1 10 39 116 77**3 6 116 195 79**3 1 195 290 95**3 4 290 369 79**
5 2 369 440 71**5 7 440 533 93**5 5 533 570 37**

The optimal solution of the problem LA15 in rdata problem
machine 1: 1 13 0 28 28**1 11 28 73 45**1 8 73 153 80**1 1 153 159 6**3 2 159 214 55**
2 12 214 268 54**2 20 268 301 33**2 8 301 357 56**2 15 357 416 59**3 6 416 499 83**2 10 499 587 88**2 5
587 627 40**4 11 627 704 77**4 16 704 756 52**4 18 756 805 49**
4 20 805 904 99**4 1 904 941 37**4 9 941 1012 71**5 3 1012 1089 77**
machine 2: 1 16 0 81 81**1 17 81 88 7**1 15 88 183 95**2 19 183 267 84**1 10 267 307 40**
1 3 307 353 46**2 13 353 426 73**3 13 426 524 98**2 9 524 615 91**3 10 615 674 59**4 17 674 762 88**3 8
762 839 77**3 7 839 919 80**3 5 919 963 44**5 18 963 990 27**4 5 990 1014 24**
5 9 1014 1031 17**5 11 1031 1089 58**
machine 3: 1 2 0 40 40**1 4 40 61 21**1 20 61 143 82**1 12 143 179 36**1 5 179 264 85**
3 11 264 272 8**2 1 272 312 40**3 1 312 393 81**3 17 393 405 12**4 4 405 430 25**
3 18 430 499 69**3 19 499 560 61**4 6 560 591 31**3 12 591 687 96**4 12 687 696 9**
3 3 696 766 70**3 14 766 793 27**4 10 793 800 7**4 3 800 855 55**4 14 855 954 99**
5 14 954 1050 96**4 7 1050 1080 30**5 2 1080 1089 9**
machine 4: 1 18 0 45 45**2 2 45 77 32**2 4 77 142 65**2 17 142 164 22**3 4 164 228 64*
*4 2 228 309 81**2 6 309 338 29**2 16 338 430 92**2 14 430 516 86**4 15 516 601 85**
4 19 601 669 68**4 13 669 761 92**5 16 761 800 39**5 10 800 880 80**5 17 880 940 60**
5 20 940 984 44**5 8 984 1081 97**5 7 1081 1089 8**
machine 5: 1 19 0 21 21**1 9 21 77 56**1 14 77 147 70**2 11 147 176 29**1 6 176 265 89**
2 18 265 358 93**1 7 358 417 59**2 7 417 455 38**3 15 455 511 56**2 3 511 576 65**
3 20 576 647 71**3 16 647 679 32**5 15 679 720 41**5 4 720 735 15**3 9 735 785 50**
5 6 785 869 84**5 12 869 879 10**4 8 879 920 41**5 13 920 1007 87**5 19 1007 1033 26**
5 1 1033 1052 19**5 5 1052 1089 37**
```

Fig. 3 LA01 and LA15 optimal solutions

Table 4 Study of the robustness of the LGO_CEA

	la25 edata	la22 vdata	la28 vdata	la30 vdata
Min	984	761	1078	1086
Max	1054	792	1107	1119
Mean	1015.4	778.55	1090.05	1100.7
Std	20.62	8.81	8.24	9.49
RSD	2.03	1.13	0.75	0.86

population of the evolutionary algorithm. As seen in Table 5, the hybrid algorithm is from 3 to 22 times slower than the CEA. The LGO_CEA realizes higher performances according to the objective function and succeeds to solve the stagnation of the search in population based metaheurists in despite of an increase in the execution time.

The hybridization of the LGO with the CEA has shown its efficiency since the results are improved for almost all the test instances. The CEA alone converges to local optima and the evolutionary search is blocked due to the presence of a dominant solution. When a dominant solution is inserted in a new generated population, the evolutionary search evolves toward better performances since it takes advantages both from the leader's characteristics and from the diversity of the population. The LGO_CEA is

Table 5 Study of execution time LGO_CEA and CEA

	Execution time			Makespan	
Instance	CEA	LGO_CEA	LGO_CEA/CEA	CEA	LGO_CEA
la21 edata	478.6	1302.57	2.72	1106	1059
la25 edata	78.1	1785.05	22.85	1052	965
la22 vdata	1359.3	10,733.85	7.89	782	755
la28 vdata	978.32	3402.57	3.47	1099	1076
la30 vdata	554.96	11,660.5	21.01	1101	1076

robust. While executing the algorithm many times on the same problem it realises near performances. The main drawback of the LGO_CEA is its execution time. The algorithm launches many times the population based metaheuristic so its execution time is higher. By comparison to the best results found in the literature, the LGO_CEA succeeds to characterise new optimal solutions and to realize whether the same or very close performances.

5 Conclusion

In this paper, a new co-evolutionary approach based on the combination of genetic algorithm with local search algorithm for solving FJSP with minimization of the makespan, is presented. The CEA applies multiple crossovers and mutations, as also as, many constructive scheduling heuristics. The algorithm selects adaptively the genetic operator to apply by favoring dynamically those realizing the best performance. The use of multiple genetic operators enhance the performance of the evolutionary process. The CEA performs better than random choice of operators. To escape local optima the CEA is combined with the leader guided optimization metheuristic. A leader is injected at the beginning of each simulation to guide the research process to a better solution. The performance of the new approach are evaluated and compared with the results obtained from other works in the literature. The effectiveness of the developed method is proved by reaching two new optimal solutions. The LGO is a generic algorithm. As a future work, it can be combined with other population based metaheuristics and evaluated on other optimization problems. Another direction is to distribute the algorithm to deal with the problem of execution time.

References

Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice-Hall, Upper Saddle River, pp 225–251

Gao J, Sun L, Gen M (2008) A hybrid genetic and variable neighborhood descent algorithm for flexible job-shop scheduling problems. Sci Direct Comput Oper Res 35(9):2892–2907

Gen M, Gao J, Lin L (2009) Multistage-based genetic algorithm for flexible job shop scheduling problem. In: Intelligent and evolutionary systems. Volume 187 of the series studies in computational intelligence, pp 183–196

Hurink E, Jurisch B, Thole M (1994) Tabu search for the job shop scheduling problem with multi-purpose machine. Oper Res Spektrum 15(4):205–215

Mastrolilli M, Gambardella LM (2000) Effective neighbourhood functions for the flexible job shop problem. J Sched 3(1):3–20

Mesghouni K, Hammadi S, Borne P (2004) Evolutionary algorithms for job-shop scheduling. Int J Appl Math Comput Sci 14(1):91–103

Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop problem. Manag Sci 42(6):797–813

Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job shop scheduling problem. Comput Oper Res 35(10):3202–3212

Roy B, Sussmann B (1964) Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS N°. 9bis, SEMA Montrouge

Singh MR, Mahapatra SS (2016) A quantum behaved particle swarm optimization for flexible job shop scheduling. Comput Ind Eng 93:36–44

Thierens D (2007) Adaptive strategies for operator allocation. Volume 54 of the series studies in computational intelligence, pp 77–90

Zhang G, Zhu H, Zhang C (2011) Hybrid intelligent algorithm for flexible job-shop scheduling problem under uncertainty. In: Mellouk A (ed) Advances in reinforcement learning, Chapter 19, pp 361–370