Flex Serv Manuf J (2011) 23:64–85
DOI 10.1007/s10696-010-9067-y

An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem

Nasr Al-Hinai · T. Y. ElMekkawy

Published online: 9 December 2010

Abstract The work presented in this paper proposes hybridized genetic algorithm architecture for the Flexible Job Shop Scheduling Problem (FJSP). The efficiency of the genetic algorithm is enhanced by integrating it with an initial population generation algorithm and a local search method. The usefulness of the proposed methodology is illustrated with the aid of an extensive computational study on 184 benchmark problems with the objective of minimizing the makespan. Results highlight the ability of the proposed algorithm to first obtain optimal or near-optimal solutions, and second to outperform or produce comparable results with these obtained by other best-known approaches in literature.

Keywords Flexible job shop · Scheduling problems · Genetic algorithms · Local search · Meta heuristic approaches

1 Introduction

Scheduling is concerned with allocating number of tasks to limited resources in order to optimize a certain performance criteria. Depending on the problem size, represented by the number of tasks and resources, the scheduling task may turn out to be the most time consuming and challenging activity within an enterprise.

Flexible job shop scheduling problem (FJSP) is a generalization of the classical job shop scheduling problem (JSP). It takes shape when alternative production routing is allowed in the classical job shop. Here, the scheduling problem becomes

N. Al-Hinai (✉) · T. Y. ElMekkawy
Department of Mechanical and Manufacturing Engineering, University of Manitoba, Winnipeg, MB R3T 5V6, Canada
e-mail: umalhinn@cc.umanitoba.ca

T. Y. ElMekkawy
e-mail: tmekkawy@cc.umanitoba.ca

more difficult to deal with as it introduces a further machine assignment decision level beside the operations' sequencing level. In manufacturing systems, most scheduling problems are very complex in nature and very complicated to be solved by conventional optimization techniques to obtain a global optimal schedule. For example, determining an efficient schedule for a flexible job shop (FJS) with n jobs and m machines will have $(n!)^m$ possible sequences (Mellor 1966). Hence, scheduling problems are considered as combinatorial optimization problems and classified as NP-hard problems (Garey et al. 1976). Nevertheless, modelling and solving the more complex FJSP is increasingly attracting the interest of many researchers. This is can be recognized by the increase in the number of research papers addressing this problem.

Due to the NP-hard nature of the FJSP, using a pure mathematical optimization approach to determine an optimal solution may not be efficient in practice. Similarly, heuristics, which can be efficient for some problem instances do not have a guaranteed distance from the optimal solution. Recently, using a high-level strategy to guide other heuristics, known as *meta-heuristics*, led to better and more appreciated results in a relatively short period. Therefore, a number of meta-heuristics were proposed in literature for the past two decades to deal with FJSP such as simulated annealing (SA), ant colony (AC), genetic algorithm (GA), etc.

The main advantage of using GA approaches in contrast of local search techniques, is the fact that GA utilizes a population of solutions in its search, giving it more resistance to premature convergence on local minima. Our approach to solve the FJSP is based on hybridising GA with an initial schedule generation heuristic and then combing it with a local search method. The current method modifies some of the already known techniques in literature and combines them to produce efficient hybridized GA architecture.

The remainder of this paper is organized as follows: Sect. 2 describes the FJSP definition. Previous research work in this area is summarized in Sect. 3. Section 4 introduces the proposed GA architecture and the local search method, respectively. The computational results are presented and discussed in Sect. 5. Finally, the research summary and possible directions for future work in this area are covered in Sect. 6.

2 Problem definition

FJSP is strongly NP-hard due to (a) assignment decisions of operations to a subset of machines and (b) sequencing decisions of operations on each machine (Tay and Wibowo 2004). The FJSP can be formulated as follows:

- There are n independent jobs of each other and indexed by i.
- All jobs are ready to start at time zero.
- Each job i has Q_i operations and the operations' sequence is given by O_{ij} for $j = 1, \ldots, Q_i$.
- There are m machines indexed by k.
- Machines never breakdown and are always available.
- For each operation O_{ij}, there is a set of machines capable of performing it represented by $M_{kij}, M_{kij} \subseteq \{1, \ldots, m\}$.

- The processing time of an operation O_{ij} on machine k is predefined and given by t_{ijk}.
- The setup time of any operation is independent of the schedule, fixed, and included in the corresponding processing time.
- A started operation cannot be interrupted (non-preemption condition).
- Each machine can process at most one operation at any time (resource constraints).
- The precedence constraints of the operations in a job can be defined for any pair of operations.

The objective is to find a schedule that has lowest possible value of makespan, where the makespan is the time required for all jobs to be processed according to a given schedule.

3 Literature review

Generally, used approaches to solve the complex FJSP can be categorized into two main basic approaches: concurrent approaches and hierarchical approaches. Hierarchical approaches are based on decomposing the problem to reduce its complexity. This complexity reduction is achieved by separating the assignment decisions level (or flexible routing) from the sequencing decisions of operations into two sub-problems. It is worth mentioning here that the sequencing decisions sub-problem is in fact a classical job shop scheduling problem. Thus, this approach had gained the interest of researchers at earlier stages like Brandimarte (1993) and Paulli (1995) both of whom solved the assignment problem using some heuristic dispatching rules and then used tabu search to solve the remaining sequencing problem. Similarly, Bona et al. (1990) followed a similar heuristic approach to handle the machine assignment part and then used simulated annealing heuristic.

On the other hand, concurrent approaches (also known as integrated approaches) integrate both problem levels and solve them simultaneously. Although this approach was attempted at early stages like in Lee and Mirchandani (1988) and in Mirchandani et al. (1988), hierarchical approach was favoured over it for sometime due to the simplifications associated with the later. However, the high quality results obtained using concurrent approach shifted the interest towards this approach to deal with the FJSP. For example, the concurrent approaches developed using tabu search methodologies as in Hurink et al. (1994), Brucker and Neyer (1998), Dauzére-Pérés and Paulli (1997), and Mastrolilli and Gambardella (2000); and simulated annealing as in Najid et al. (2002).

Among different meta-heuristic algorithms, GA is considered to be a very successful to tackle the FJSP and this can be noticed by the growing number of papers discussing this topic. Falkenauer and Bouffouix (1991) were among the first to propose a machine parallel representation to solve JSP. They encoded the chromosomes using list of machines operating in parallel. Their work was then extended by other researchers like Mesghouni et al. (1997) who proposed parallel job representation and Chen et al. (1999) who divided the chromosome into two strings; A and B; where the first is defining the routing policy and the second defines the sequence of operations on each machine.

Similarly, Ho and Tay (2004) proposed a new GA structure called GENACE. Their chromosome consists of two parts, one dedicated to define the operation order and the second for machine selection. GENACE was then combined with a learning knowledge procedure and converted to another GA structure called LEGA by Ho et al. (2007). A similar chromosome representation was adapted by Gao et al. (2008) who developed a GA hybridized with variable neighborhood descent local search procedures for the flexible job shop problem.

Kacem et al. (2002a, b) suggested using an assignment table representation of chromosomes. In the assignment table, a machine is mapped to a consequent operation. This work was recently modified by Pezzella et al. (2008) by integrating more strategies in the genetic framework that led to better results. Kacem (2003) used a task sequencing representation of chromosomes where each cell represents an operation on a machine. Giovanni and Pezzella (2010) proposed an improved genetic algorithm for the distributed and flexible job shop scheduling problem. Wei and Qiaoyun (2009) offered an adaptive genetic algorithm that considers the processing time, the completion time of previous operation and the idle time of current machine to select a suitable machine in the decoding process when solving the FJSP.

Hussain and Joshi (1998) proposed a two pass GA for the job shop scheduling problem with alternate routing. Yang (2001) offered GA-based discrete dynamic programming approach. Jia et al. (2003) proposed a modified GA to solve distributed scheduling problems. Chan et al. (2006) introduced the idea of dominant genes, which is based on the principle of Automata introduced by White and Oppacher (1994). Zribi et al. (2007) used a hierarchical GA approach to solve the machine assignment and scheduling of FJSP. Girish and Jawahar (2008) proposed two concurrent meta-heuristic approaches, genetic algorithm (JSSGA) and ant colony (JSSANT) to solve job shops with multiple routings. Xu et al. (2009), Xing et al. (2010) and Ling et al. (2010) presented an ant colony optimization algorithm for the FJSP.

Xia and Wu (2005) combined particle swarm optimization algorithm with simulated annealing to form hybrid approach for the multi-objective FJSP. Girish and Jawahar (2009) offered particle swarm algorithm to solve FJSP. Also, Zhang et al. (2009) proposed a hybridized particle swarm optimization algorithm with tabu search to solve multi-objective FJSP. Liu et al. (2009) formulated a multi-particle swarm approach to solve multi-objective FJSP. Xing et al. (2009) addressed a local search method based on empirical knowledge for the multi-objective FJSP. Another local search method that adapts the concept of climbing discrepancy search method is proposed by Hmida et al. (2010).

4 GA structure

4.1 Chromosome coding

A proper chromosome representation has a great impact on the success of the used GA. Cheng et al. (1996) gave a detailed tutorial survey on papers using different GA chromosome representations to solve classical JSP. It can be concluded from their work (and others like Ho et al. 2007; Mattfeld 1996; and Tay and Wibowo 2004)

(a) 221–131–111–212–322–223–332

(b) 121–131–111–212–322–223–332

Fig. 1 **a** Chromosome coding; **b** alternative routing

that the search space of an operation-based representation covers the whole solution space and any permutation of operators can correspond to a feasible schedule. For a recent overview discussing these aspects readers are referred to Hart et al. (2005). In light of the above, the current work uses a similar effective permutation-based chromosome representation used in Kacem et al. (2002a, b), Kacem (2003) and Chan et al. (2006). The used representation composes of a string consisting of triples (k, i, j) for each operation forms the chromosome, where

- k is machine assigned to the operation;
- i is current job number;
- j is the progressive number of that operation within job i.

This chromosome representation integrates the machine assignment decision level and sequence decision level in one simple gene representation. Such a representation reduces the memory usage and permits more efficient genetic operators to be considered (see Sect. 4.5).

Another advantage of this representation is its ability to model alternative routing of the problem by changing the index k. The length of the chromosome is equal to the total number of operations (*tot_noper*) to be scheduled. Figure 1 shows a sample encoding of a chromosome for FJSP with three jobs and three machines according to the processing times given in Table 1. In Fig. 1a, the first gene (221) characterizes that operation 1 of job 2 is assigned to machine 2. Since this operation can be carried out on another machine, machine 1, then the gene can be modified to (121) as shown in Fig. 1b. The scheduling priority of operations is set to be from left to right. Based on this priority, a chromosome is decoded to produce an active schedule (see Sect. 4.2). Furthermore, both FJSP sub-problems known as total FJSP (T-FJSP) and partial FJSP (P-FJSP) can be represented with this representation without any modification (T-FJSP and P-FJSP are addressed in Sect. 5).

4.2 Chromosome decoding

Even though the used chromosome representation or *genotype* corresponds to feasible solution (schedule), mapping it to a proper *phenotype* is essential to reduce the very large feasible solution-space. Feasible schedules are categorized into semi-active schedules, active schedules and non-delay schedules. French (1982) and Pinedo (2002) demonstrated that the set of non-delay schedules is a sub-set of the active schedules which is a sub-set of the semi-active schedules. They also verified that for regular performance measures an optimal solution exists within the set of active schedules. Accordingly, our decoding algorithm minimizes the solution-space by constructing active schedules, while still ensuring that an optimal solution can be found. The decoding algorithm to produce active schedule is shown in *Algorithm Decode*.

Table 1 Processing times for a FJSP with 3 jobs and 3 machines

J	O	M1	M2	M3
J_1	O_{11}	2	–	4
	O_{12}	–	3	–
J_2	O_{21}	2	3	–
	O_{22}	4	3	5
	O_{23}	–	1	–
J_3	O_{31}	2	3	1
	O_{32}	1	2	3

Algorithm Decode (*decoding a chromosome to an active schedule*)

```
1.  Initialize Gantt chart structure
2.  For each gene reading from left to right do
3.      Identify the operation $O_{i,j}$
4.      Identify the machine $M_k$ which processes $O_{i,j}$ from the first gene's digit and its
        processing time $t_{i,j,k}$
5.      If $O_{i,j}$ is the first operation Then
            Set $t_0$ to 0
6.      Else
            Set $t_0$ to be stop time of the predecessor operation $O_{i,j-1}$
7.      End If
8.      If $M_k$ did not process any operation Then
            Set $t_1$ to 0
9.      Else
            Set $t_1$ to be the stop time of the last operation on $M_k$
10.     End If
11.     If $t_1 \leq t_0$ Then
            Add $O_{i,j}$ to $M_k$ starting at $t_0$
12.     Else If it exist between $t_0$ to $t_1$ (time interval between two consecutive operations on
        $M_k$) $\geq t_{i,j,k}$ Then
            Add $O_{i,j}$ to $M_k$ starting at the end of the finished processing time of the
            operation to the left
13.     Else
            Add $O_{i,j}$ to $M_k$ starting at $t_1$
14.     End if
15. End for
```

4.3 Initial population

The efficiency of meta-heuristic algorithms that use initial solutions as a starting point can be improved by generating proper solutions at promising points on the solution-space. However, if all initial solutions are generated and tuned to satisfy the objective function, then the risk of the algorithm being trapped at local minima increases. To satisfy these two conflicting requirements, our genetic search process adopts two tactics to generate the initial population.

The first procedure is to randomly generate a certain percentage of chromosomes. Each chromosome's gene is generated first by randomly selecting a progressive operation number of an unscheduled operation and then by randomly assigning that operation to an appropriate machine. This procedure is repeated until all operations are coded.

The second procedure uses a heuristic approach to construct a Gantt chart which is used to generate a chromosome. The procedure considers the processing time as well as the work load on the machine while assigning an operation. Thus, an operation may not necessarily be assigned to the machine with minimum processing time, but it will be assigned to the machine that will finish it sooner than other appropriate machines. The pseudo-code of the developed heuristic is shown in *Algorithm Ini-PopGen*. It should be emphasized that this algorithm generates active schedules.

Algorithm Ini-PopGen (*initial population generation to form an active schedule*)

```
1.  Initialize Gantt chart structure
2.  Generate a random job order array $J$ (e.g. $J_2\ J_1\ J_3$)
3.  Set $oper\_count = 1$and $oper = 1$
4.  While ($oper\_count \le tot\_noper$) do
5.      For jobs $i \in J$ do
6.          Find all proper machines $M$ that can process operation $O_{i,oper}$ (e.g. $M_1\ M_3$)
7.          Generate an $O_{i,oper}$ stopping time array $T$ of the same size of $M$
8.          Initialize $t_0$ and $t_1$
9.          If $O_{i,oper}$ is the first operation Then
10.             Set $t_0$ to 0
11.         Else
12.             Set $t_0$ to be stop time of the predecessor operation $O_{i,oper-1}$
13.         End If
14.         For machines $k \in M$ do
15.             Identify processing time $t_{i,oper,k}$ of $O_{i,oper}$ on machine $M_k$
16.             If $M_k$ did not process any operation Then
17.                 Set $t_1$ to 0
18.             Else
19.                 Set $t_1$ to be the stop time of the last operation on $M_k$
20.             End If
21.             If $t_1 \le t_0$ Then
22.                 Add $O_{i,oper}$ to $M_k$ starting at $t_0$ and set $T_{oper,k} = t_0 + t_{i,oper,k}$
23.             Else If it exist between $t_0$ to $t_1$ (time interval between two
                consecutive operations on $M_k$) $\ge t_{i,oper,k}$ Then
24.                 Add $O_{i,oper}$ to $M_k$ starting at the end of the finished processing
                    time of the operation to the left and set $T_{oper,k}$ equal to this
                    time + $t_{i,oper,k}$
25.             Else
26.                 Add $O_{i,oper}$ to $M_k$ starting at $t_1$ and set $T_{oper,k} = t_1 + t_{i,oper,k}$
27.             End If
28.         End For
29.         Find the min. stopping time on $T$ and assign $O_{i,oper}$ to that machine. If more
            than one machine satisfy this, Then assign $O_{i,oper}$ to the first machine with
            the min. stopping time on $T$
30.         Encode the selected machine $M$, job $J_i$, and operation $oper$ into the
            chromosome
31.         $oper\_count$ ++
32.     End For
33.     $oper$ ++
34. End While
```

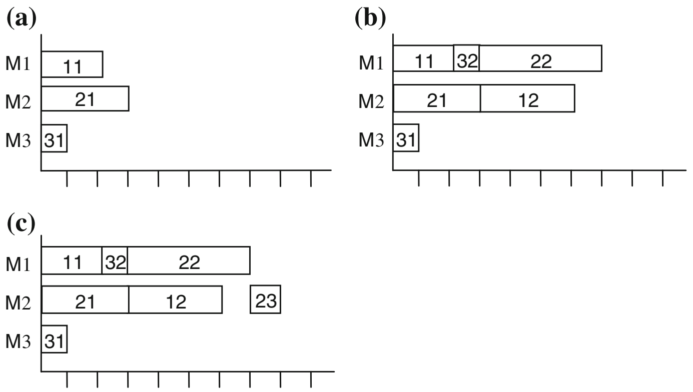


Fig. 2 A few steps of the Ini-PopGen algorithm

Figure 2 demonstrates how Ini-PopGen algorithm creates an active schedule for the data given in Table 1. Suppose the randomly created jobs priority is [3, 1, 2] (step 2). Therefore, the first operations of jobs 3, 1, and then 2 will be scheduled in the Gantt chart as shown in Fig. 2a, respectively (steps 4–34). Fig. 2b shows the scheduling of the second operations of the jobs. If we consider for example O_{22}, then it can be noted that the operation was assigned to M_1 even though the processing time on machine M_2 is the smallest among the other machines. However, M_1 is selected as it is the machine that can finish processing this operation before others (Step 29). The final Gantt chart is shown in Fig. 2c, which indicates that it is the optimal schedule for this machines' assignment as the operations of job 2 falls on the critical path of this schedule and they cannot be finished sooner without violating the precedence constraints. This emphasizes the strength of this algorithm in obtaining good initial schedules or chromosomes.

4.4 Selection

Selection is an important element in GA. The task is to select individuals for reproduction by moving them into a mating pool. Our selection method of individuals for mating is divided into two phases:

1. Phase 1 (forming the *donors' mating-pool*): Roulette wheel technique is chosen to be the main driving motor in selecting the donor chromosomes and form what we wish to term as, 'the donors' mating-pool'. The donors' mating-pool is formed at the beginning of each evolution process before the crossover stage starts. Two procedures were tested in this phase. The first procedure starts by first linearly ranking the individuals and sorting them according to their fitness values. Then, roulette wheel is used to select ranked individuals based on a selection probability given by the equation:

$$P_s = \frac{F_{ind}}{F_{tot}}, \quad ind = 1, \ldots, N \tag{1}$$

where, P_s is the probability of choosing the *ind*th individual; N is the population size; F_{ind} is the *ind*th individual fitness; and F_{tot} is the total fitness of all individuals in the current generation.

The second procedure is to apply roulette wheel to form the donors' mating-pool using Eq. 1 without ranking the individuals. After comparing the two methods, our results show that the second procedure performs better than the first one. This could be related to the population diversity that results from both procedures. When individuals are ranked according to their fitness values this gives individuals with better fitness values higher chances to be selected to form the mating pool and have higher chances to reproduce. Though such outcome is preferred to enhance the exploration around promising areas of the solution-space, at a certain stage of the evolution process, this causes the algorithm to converge prematurely to a local optimum. This is avoided in the second procedure by giving individuals with less fitness values the chance to be part of the evolution process and reduce the chances of the algorithm being trapped in local minima.

2. Phase 2 (forming the *receivers' mating-sub-pool*): After the donor's mating-pool has been formed and at the beginning of each crossover process, individual in the donors' mating-pool are subjected to a crossover probability P_c. If the individual satisfies this probability, then Phase 2 in the selection procedure starts by forming what we term as 'receivers' mating sub-pool' using an n-Size tournament method, otherwise the donor will form a child. Phase 2 procedure starts by selecting n different random number of chromosomes from the population and moves them to the receivers' mating-sub-pool. Following on, the individuals within the sub-pool are ranked according to the fitness and the best is chosen for reproduction. It is worth mentioning here that the donor is prevented from being reselected in this procedure.

4.5 Genetic operators

The performance of genetic algorithms to a great extent depends on the performance of the genetic operators used (Gen and Cheng 2000). Therefore, designing the right genetic operators is very essential for the success of any GA. Furthermore, when applying genetic operators; crossover and/or mutation; there is a high chance of forming infeasible chromosomes by, for example, violating the precedence constraints among operations of the same job. In such cases, a repair or correction mechanism has to be implemented on the infeasible offspring. Such repair procedure is time-consuming. Therefore, it is more practical to design the operators to respect such precedence constraints. Ho et al. (2007) and Gao et al. (2008) satisfied this by dividing their chromosome into two parts where the first defines the operation order and the second defines the machine selection. Their genetic operators were separately applied to each part. Therefore, they were able to avoid producing infeasible chromosomes. However, such chromosome representation complicates the GA architecture and requires that the genetic operators to independently be applied to each part, which in turn increases the required computational time.

Since our algorithm implements a chromosome representation that integrates both FJSP levels by adopting a gene of three triples (machine, job, operation), a genetic operator which satisfies the previous requirement has to be carefully designed otherwise a repairing technique will be a necessity. In order to satisfy this, *Precedence Preserving Order-based Crossover* (POX) (Kacem et al. 2002a) and a modified *Position Based Mutation* (PBM) (Mattfeld 1996) as well as *Machine Based Mutation* (MBM) are selected as the genetic operators.

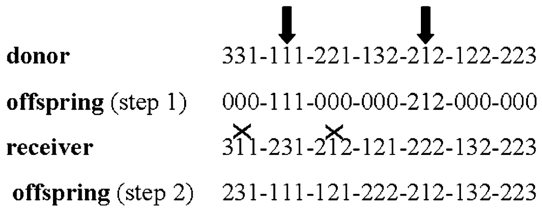
4.5.1 Crossover operator

The idea in POX is to transfer the selected operations in the donor to the same position in the offspring. Furthermore, we have modified the POX operator so that it does not treat the parents symmetrically. This modification assists the algorithm to control the spread of the genetic information and keep the *genetic drift* to minimum. The crossover is implemented in two steps. In step 1, a gene is randomly selected from the donor and all genes containing operations belonging to the job found in that gene are copied to the offspring. Meanwhile, the corresponding operations in the receiver are located and deleted. In step 2, the remaining offspring's genes are completed by moving the remaining operations from the receiver and inserting them in the offspring's empty genes in the same order. Figure 3 exemplifies this procedure for two randomly created chromosomes representing the 3 $\times$ 3 problem given in Sect. 4.1. The selected genes from the donor have been pointed to by an arrow, and the corresponding deleted operations in the receiver are marked by ($\times$). It can be observed from Fig. 3 that POX includes an implicit mutation. For example, consider the relative position of operation 2 of job 2 to operation 2 of job 1. In both parents, operation 2 of job 1 precedes operation 2 of job 2. However, this is not the case in the offspring. Therefore, if both operations are to be processed on the same machine, then unlike the parents, operation 2 of job 2 will be scheduled before operation 2 of job 1 in the offspring. Moreover, when implementing POX the operations belonging to the same job are moved from the parent to the offspring in the same order. Consequently, the precedence constraints of jobs are not violated and hence no infeasible chromosome is produced. Thus, no repairing mechanism is required.

4.5.2 Mutation operator

After creating a number of children equal to the number of parents using crossover according to a crossover probability P_c, children are subjected to mutation

Fig. 3 POX example


donor 331-111-221-132-212-122-223
offspring (step 1) 000-111-000-000-212-000-000
receiver 311-231-212-121-222-132-223
offspring (step 2) 231-111-121-222-212-132-223

according to a mutation probability P_m. This is done to maintain the diversity of the chromosomes and to introduce some extra variability into the population. Two strategies were followed during the mutation process:

1. The first strategy called *Machine Based Mutation* (MBM), where a random number of operations (denoted as *nrand*) are selected and reassigned to another machine. As introduced earlier, the purpose of mutation is to increase the diversity of the population. However, if a drastic change is imposed in the chromosomes' structure, then the genetic search will turn to be a random search. Therefore, the resultant chromosome from mutation should not be significantly different from the original chromosome. Number of experiments were conducted to select the appropriate values of *nrand* that satisfy the previous condition. Hence, *nrand* is limited between [1, 3]. After *nrand* is uniformly selected from that range, *nrand* operations are randomly selected within the child's chromosome. The selected operations are then reassigned to another machine if applicable to form a *child'*. In order to reduce the computational time, a local memory is linked to each operation. This local memory contains information about the machines that can process this specific operation and their number. This mutation procedure is illustrated in Fig. 4, where it is assumed that the generated *nrand* = 2 and the randomly selected genes are pointed to by an arrows. Figure 4a expresses that only one of the two operations can reassigned to another machine.
2. The second strategy called modified *Position Based Mutation* (PBM) (Mattfeld 1996). PBM was originally designed for JSP using single triple permutation-based chromosomes representation. Thus, the PBM is modified so that no infeasible chromosomes are produced. This mutation starts by randomly selecting an operation from *child'* and reinserting it at another position in the new child. Then, the remaining operations are copied from *child'* to the new child taking care of the precedence constraints of the moved operation. Figure 4b demonstrates this mutation procedure, where the selected gene is underlined and the new position is pointed to by an arrow.

After crossover and mutation is implemented, the children are taken to form the new generation. The algorithm terminates after reaching a maximal number of generations.

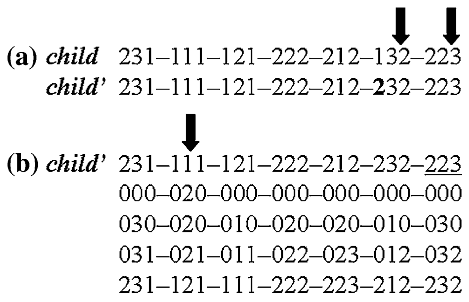

(a) child 231–111–121–222–212–132–223
child' 231–111–121–222–212–232–223
(b) child' 231–111–121–222–212–232–223
000–020–000–000–000–000–000
030–020–010–020–020–010–030
031–021–011–022–023–012–032
231–121–111–222–223–212–232

Fig. 4 **a** Sample procedure of MBM; **b** sample procedure of modified PBM

4.6 Local search

The use of local search techniques has been proven to be useful in solving combinatorial problems. Local search methods are applied to a neighborhood of a current solution. In the case of JSP, a neighborhood is achieved by moving and inserting an operation in a machine sequence. Therefore, when designing a local search method that works on neighborhood of solutions, two main issues have to be considered. The first issue is that the size of the neighborhood has to be limited. The second issue is the feasibility of solutions.

Generally, there are two scenarios to deal with the feasibility issue. The first scenario is to design a local search that only considers feasible moves as in the two neighborhood functions proposed by Mastrolilli and Gambardella (2000). The second scenario is to design a local search that accepts all moves and then either rejects the infeasible moves or implement a repairing procedure like the proposed repairing technique presented by Murovec and Šuhel (2004).

Brandimarte (1993) suggested the following neighborhood structures:

1. Neighborhood $\mathcal{N}_1$ where a sample size is set to randomly sampling the potential exchanges within the neighborhood.
2. Neighborhood $\mathcal{N}_2$ where a job is randomly selected and it is exchanged on each machine with the adjacent jobs in each machine sequence.
3. Neighborhood $\mathcal{N}_3$ where a machine is randomly selected and the complete set of job exchanges on that machine is considered.
4. Neighborhood $\mathcal{N}_4$ where operations on the critical path are considered for the exchange.

The second and third neighborhood structures limit the search space by confining it to either focus on a specific arrangement, whereas the first gives some degree of random exploration. Several researchers have verified that when the makespan is the considered objective function, then neighborhood $\mathcal{N}_4$ is very useful since operations not laying on the critical path do not affect the makespan (see for example Brandimarte 1993; Mastrolilli and Gambardella 2000; Murovec and Šuhel 2004). However, a local search that is solely devoted to work on critical path heads directly toward deterministically predictable local minima. On the other hand, a local search that is based on random search helps preserving the diversification of the population. Hence, a local search that combines both approaches by introducing randomness in a local search that is related to the critical path may be more desirable. Therefore, we propose using a local search method that combines neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_3$ while defining hill climbing heuristic that works on blocks of moves that contain a critical block as subset (for more details about critical blocks, readers are referred to Mattfeld 1996).

The proposed local search starts with a feasible schedule $\mathcal{S}$ as an input. The input schedule is set to $\mathcal{S}_{best}$ which stands for the best found solution. Then, a neighborhood consisting of blocks containing feasible moves on each machine is constructed. A feasible move is defined by two consecutive operations on the same machine that can be interchanged without violating the precedence constraint.

Therefore, each block will contain at least two consecutive operations which correspond to one move. To that, an ordinary hill climbing heuristic is applied to bring the new schedule $\mathcal{S}$' to the local optimum with respect to the used neighborhood. The hill climbing heuristic works by interchanging or swapping two consecutive operations within one block at a time. Every time a move is done, the new makespan of $\mathcal{S}'$ is estimated. If the new $\mathcal{S}'$ is better (i.e. has a lower makespan) than $\mathcal{S}_{best}$, then $\mathcal{S}_{best}$ is replaced by $\mathcal{S}'$. The procedure is repeated until a maximal number of iterations (*loc_iter*) without improving moves is reached. The pseudo-code of the local search heuristic is shown in *Algorithm LS.*

Algorithm LS (*local search)*

```
1.  Calculate the performance $C_{max}(\mathcal{S}_{best})$ of the current schedule $\mathcal{S}$
1.  Set count to 1
2.  While ((count ≤ loc_iter) && ($\mathcal{S}' \in \mathcal{N}_{hc,feasible}(\mathcal{S}_{best})$)) do
3.          If $C_{max}(\mathcal{S}') < C_{max}(\mathcal{S}_{best})$ Then
4.                  Update $\mathcal{S}_{best}$ by setting $\mathcal{S}_{best} = \mathcal{S}'$
5.                  Set count to 1
6.          Else
7.                  Count++
8.          End If
9.  End While
```

To illustrate how the local search works, consider the Gantt charts shown in Fig. 5. Assuming that Fig. 5a represents an input schedule $\mathcal{S}_{best}$ to the local search, blocks; which may or may not include critical operations; containing feasible moves are in machine 1 (O_{32}, O_{22}, O_{11}), and in machine 2 (O_{23}, O_{12}). It can be seen that O_{21} is not included in the second block as if it is to be swapped with O_{23}; it will result in an infeasible schedule. Since machine 3 does not process any operation other than O_{31}, then no block is formed. The first move is done by swapping O_{32} and O_{22}. However, since in our algorithm we consider active scheduling this move does not change the schedule. The second move is done by interchanging O_{22} and O_{11}. This produces a better schedule $\mathcal{S}'$ as shown in Fig. 5b and hence, $\mathcal{S}_{best}$ is replaced by $\mathcal{S}'$. In the same time, the blocks are automatically updated with the new changes in the schedule and the modified blocks become in machine 1 (O_{32}, O_{11}, O_{22}), and in machine 2 (O_{12}, O_{23}). Since an improvement was found in the previous move, the algorithm continues by moving operations on the second block (O_{12}, O_{23}). Again, since active schedules are considered, this move will result in no improvement of the makespan of the current schedule. Assuming that the maximum number of allowed moves without improvement is not reached, the algorithm continues the search by considering moves within block (O_{32}, O_{11}, O_{22}). Interchanging O_{32} and O_{11} results in an improved schedule as shown in Fig. 5c and again $\mathcal{S}_{best}$ is replaced by $\mathcal{S}'$. The algorithm iterates until the termination criterion is met.

In order to reduce the computation time, *loc_iter* is limited and rather kept small. However, this is compensated for by combining the local search with the GA many

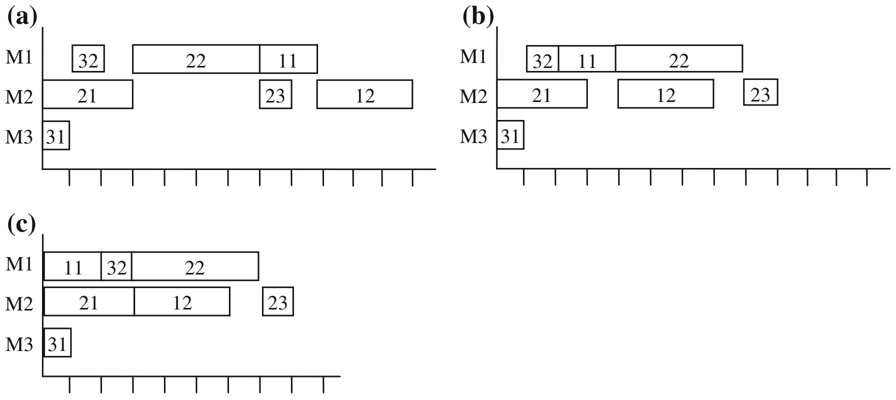


Fig. 5 Gantt charts of three neighborhoods

times during the optimization process and every time the local search is called there is a different starting solution. Hence, it is designed for exploitation purposes whereas GA takes care of the exploration of the solution space. Furthermore, unlike other memetic algorithms, the local search is only applied at every *d*th generations of the evolution process. This is to allow the population to have some diversity and enhancing the chances of preventing the solution from being trapped on local minima.

4.7 Elitism strategy

To prevent the loss of the genetic information of the current best chromosome during the evolutions, a global memory containing the elitism is generated. During the evolution process (crossover or mutation) both best and worst chromosomes are identified and recorded. At the end of each generation, the best chromosome is compared with the elitism. If the best chromosome has a better fitness value than the elitism, then the global memory is updated by replacing the elitism with this new chromosome. On the other hand, the global memory does not interfere with the genetic evolution except if the fitness value of the best chromosome is worse than that of the elitism. In such case, the elitism is used to replace the worst chromosome in the population. However, this replacement procedure takes place every *k*th generation. This strategy enables chromosomes to naturally evolve in almost all generations and hence, allowing an acceptable level of diversity on the population. Meanwhile, the knowledge gained from all generations is accumulated and only used to guide the evolution process when necessary.

5 Computational results

FJSP are classified into two sub-problems known as total flexible job shop problems (T-FJSP) and partial flexible job shop problems (P-FJSP). In T-FJSP each operation can be processed on any machine of *M*. On the other hand, in P-FJSP each operation can be processed on at least one machine of a subset of *M*. Therefore, several sets of problem instances have been considered.

1. The first set (KMData) consists of 6 instances, 3 T-FJSP taken from Kacem et al. (2002b); 1 T-FJSP taken Mesghouni et al. (1997), 1 P-FJSP taken from Lee and DiCesare (1994); and 1 P-FJSP taken Kacem et al. (2002a).
2. The second set (BRData) consists of 10 P-FJSP instances taken from Brandimarte (1993).
3. The third set (BCData) consists of 21 P-FJSP instances taken from Barnes and Chambers (1996).
4. The fourth set (DPData) consists of 18 P-FJSP instances taken from Dauzére-Pérés and Paulli (1997).
5. The fifth set (HUData) consists of 129 P-FJSP instances created from 43 classical job shop instances divided into three subsets, EData, RData and VData taken from Hurink et al. (1994).

The performance of the different components of the proposed hybrid algorithm is analysed using all sets.

For the hGA, experiments were conducted on an Intel® Pentium® D CPU 2.8 GHz with 2 GB RAM. For each test problem, a set of 5 runs were performed. A number of experiments were conducted for the same test problem in order to determine the hGA parameters. Due to space limitation these experiments cannot be fully detailed. However, we adopted the following procedure to identifying the values for each parameter. First, we randomly selected number of test cases from each of the five set instances. For each selected problem we fixed the values of all hGA parameters and only varied the value of the parameter under consideration. For example, to determine the value of the crossover probability, the parameters for the evolution were fixed to: mutation probability = 0.3, number of maximal generations = 1,000; maximum number of moves without improvement in the local search $loc\_iter = min\ [tot\_noper, 175]$; the worst chromosome is replaced by the elite chromosome every $k = 3$ generations; number of parents in the receivers' mating sub-pool = 4; number of generations to perform local search (*d* value) 10, and the population size = 200. Then crossover probability of 0.9, 0.7, and 0.5 were investigated by solving the test problem 5 times. After solving all randomly selected test problems, the crossover probability that performed better in terms of solution quality is selected for the remaining instances. A similar procedure is followed in determining the values of remaining other parameters. In conclusion of the previous analyses, the following parameter values are common in all test cases: crossover probability 0.7; mutation probability 0.3; number of maximal generations 1,000; maximum number of moves without improvement in the local search $loc\_iter = min\ [tot\_noper, 175]$; the worst chromosome is replace by the elite chromosome every $k = 3$ generations; number of parents in the receivers' mating sub-pool 4 for the KMData and BRData sets, and 6 for the rest; and number of generations to perform local search (*d* value) as well as the population size are reported in Table 2.

The first two sets are solved by using initial population generation heuristic (Ini-PopGen), initial population generation heuristic with local search procedure (Ini-PopGen + LS), and finally using the full hybridised algorithm (hGA). In each

Table 2 *d* Value and population size used in test sets

Test set	Instance		*d* Value	Population size
KMData	All		10	200
BRData	MK1/6/7/10		30	1,200
	All remaining		10	200
BCData	All		30	1,500
DPData	All		30	1,200
HUData	EData	mt6/10	20	200
		mt20	30	800
		la01–la13	20	400
		la14–la25	20	500
		la26–la40	30	1,200
	RData	mt6/10/20	20	300
		la01–la25	20	400
		la26–40	30	1,000
	VData	mt6/10/20	10	200
		la01–la40	10	300

instance of these sets, 60 instance schedules are created using the Ini-PopGen and another 60 schedules are created using Ini-PopGen + LS. The best and the average makespan of 5 runs are compared. Table 3 shows the results obtained for KMData set and compares them with these obtained by Ho et al. (2007) and Gao et al. (2008). The values in bold-face identify the best values obtained for each instance using all algorithms. The values between braces are the average results. It can be seen from Table 3 that our proposed algorithms are very efficient in solving all instances. Ho et al. (2007) solved these instances using two methods. Firstly they used composite dispatching rules population generation (CDR-PopGen), and in the second they used LEarnable Genetic Architecture (LEGA). Our proposed Ini-PopGen is able to obtain the minimum known results for all instances except for Ex4. It even outperformed the best and the average results obtained by CDR-PopGen in Ex3 and Ex6; and the average results obtained by LEGA in Ex3 and Ex4. In Ex3 proposed by Mesghouni et al. (1997); Ini-PopGen was able to obtain the optimal results of 7 time units. The optimal result was also obtained by Mesghouni et al. (1997) using GA, but after 1500 generations. Furthermore, results in Table 3 indicate that Ini-PopGen can handle small and medium T-FJSP instances. The P-FJSP instance Ex5 consists of 5 jobs, 3 machines and 4 operations in each job with a lot size of 10. This problem was solved by Lee and DiCesare (1994) using Petri Nets combined with heuristic search and obtained an average makespan of 439; Kumar et al. (2003) using Ant Colony approach and obtained an average makespan of 420; and by Chan et al. (2006) using GA with dominant genes (DGA) and obtained minimum makespan of 360 with an average of 374. It can be noticed from Table 3 that the proposed algorithms outperformed the previous results in both minimum and average makespans. Figure 6 shows the Gantt chart for the production scheduling of this example. For Ex6 taken from Kacem et al. (2002a), our algorithms outperform that

Table 3 Comparison with other meta-heuristics on KMData

Reference	Problem size	G, S and G (2008)	Ho et al. (2007)		Proposed methods		
			CDR-PopGen	LEGA	Ini-PopGen	Ini-PopGen + LS	hGA
Kacem et al. (2002b)	**Ex1** 4 × 5	–	**11(11)**	**11(11)**	**11(11)**	**11(11)**	**11(11)**
	Ex2 10 × 7	–	**11(11)**	**11(11)**	**11(11)**	**11(11)**	**11(11)**
Mesghouni et al. (1997)	**Ex3** 10 × 10	**7(7)**	8(8)	**7**(7.56)	**7(7)**	**7(7)**	**7(7)**
Kacem et al. (2002b)	**Ex4** 15 × 10	**11(11)**	12(12.24)	12(12.04)	12(12)	12(12)	**11**(11.2)
Lee and DiCesare (1994)	**Ex5** 5 × 3	–	–	–	**350**(352.5)	**350**(351)	**350(350)**
Kacem et al. 2002a	**Ex6** 8 × 8	**14(14)**	16(16)	**14(14)**	**14**(14.5)	**14**(14.2)	**14(14)**

Values written in bold are the best values

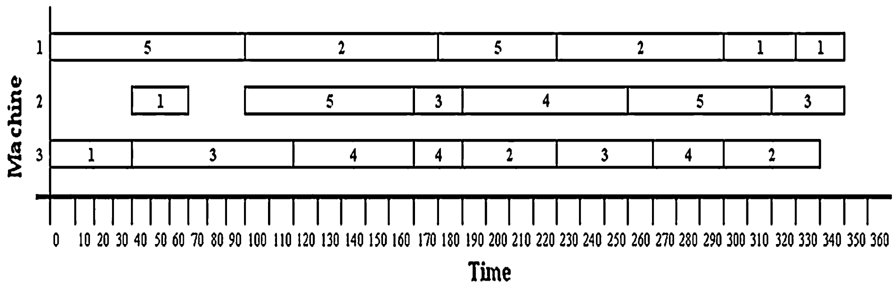


Fig. 6 Diagram of Gantt chart for the production scheduling of Ex5

proposed by Ho et al. (2007) as the Ini-PopGen is able to obtain the best known minimum makespan. Moreover, the best results obtained by hGA for Ex3, Ex4 and Ex6 are the same as these obtained by Gao et al. (2008).

In Table 4 we compare our results on the BRData set with these obtained by Ho et al. (2007) using CDR-PopGen and LEGA; and these obtained by Girish and Jawahar (2008) who solved them using Ant Colony Optimization (JSSANT) and Genetic algorithms (JSSGA). Table 4 shows that the proposed hGA either outperforms other algorithms or obtain the same minimum makespan in all cases. Furthermore, Table 4 also highlights that the proposed Ini-PopGen has a very acceptable efficiency in solving P-FJSP as it is able to produce comparative results with these obtained by other best methods of JSSANT and LEGA. For example, Ini-PopGen is outperforming JSSGA in seven out of ten MK instances; and JSSANT in instances MK06, MK09 and MK10. Moreover, the noticeably close values of the obtained results between the minimum and the average makespan in both benchmark sets (T-FJSP, and P-FJSP) highlights the repeatability of the proposed hGA and its consistency in obtaining optimal or near optimal result.

When comparing the performance of Ini-PopGen algorithm with the performance of Ini-PopGen combined with LS (Ini-PopGen + LS) in Tables 3 and 4, it can be observed that no improvements are achieved in all T-FJSP and, in general,

Table 4 Comparison with other meta-heuristics on MKData

Problem size	Girish and Jawahar (2008)		Ho et al. (2007)		Proposed methods		
	JSSANT	JSSGA	CDR-PopGen	LEGA	Ini-PopGen	PopGen +LS	hGA
MK01 10 × 6	**40(40)**	**40**(41.2)	42(42)	**40**(41.5)	42(42)	42(42)	**40(40)**
MK02 10 × 6	**26**(26.8)	**26**(29.6)	30(30)	29(29.1)	28(28.88)	28(28.8)	**26**(27.10)
MK03 15 × 8	**204(204)**	212(215.2)	–	–	**204(204)**	**204(204)**	**204(204)**
MK04 15 × 8	66(66.8)	71(74)	68(68)	67(68.82)	74(74.75)	73(74)	**61(62.83)**
MK05 15 × 4	174(176.4)	188(191.2)	179(179.3)	176(178.1)	179(179.63)	177(179.4)	**173(174.67)**
MK06 10 × 15	77(78.4)	81(83)	69(69.2)	67(68.82)	70(70.63)	69(70)	**62(64.83)**
MK07 20 × 5	143(143.8)	152(154)	153(153.88)	147(152.9)	150(153.75)	150(153.4)	**141(143)**
MK08 20 × 10	**523(523)**	533(545)	527(528.44)	**523**(523.34)	544(544.25)	**523**(524.2)	**523(523)**
MK09 20 × 10	328(341.2)	378(382)	326(328.78)	320(327.74)	326(326)	319(323.2)	**307(307)**
MK10 20 × 15	247(254.8)	265(281.4)	234(236.12)	229(235.72)	233(234)	233(234)	**214(218.33)**

Values written in bold are the best values

improvements in P-FJSP instances are small. This can be related to two main reasons. The first reason can be related to the possibility that the obtained solution using Ini-PopGen algorithm is the minimum or near minimum feasible solution. This reason can be supported by considering results obtained using different algorithms in Table 4, where different algorithms obtained the same minimum solutions. The second reason may be related to the nature of FJSP where it consists of two decision levels. Here, it is possible that solutions produced using Ini-PopGen are optimized to a near optimal local solution for that machines' assignment decision level. In such case, expected improvements using any local search method that only works on the sequencing decision level, like the proposed LS, are small. This conclusion can be sustained by the obtained results of instances MK08 and MK09 where the LS has much improved the results obtained by Ini-PopGen and it even brought the solution of MK08 to optimality.

Table 5 compares the best results obtained by our hGA to the best results of the particle swarm algorithm proposed by Girish and Jawahar (2009); and GA algorithms proposed by Zribi et al. (2007); Pezzella et al. (2008); and Gao et al. (2008), on the BRData set instances. The first column reports the instance name, the second reports the best-known lower bound and the third reports our best obtained results. The remaining columns report the best results of the three algorithms with the relative deviation with the respect to our algorithm defined as follows:

$$dev = [(MS_{comp} - MS_{hGA})/MS_{hGA}] \times 100 \qquad (2)$$

where MS_{comp} is the makespan we compare to and MS_{hGA} is the makespan obtained by our algorithm. The results show that our results are either outperforming other methods or of comparable quality.

In Table 6 we compare the relative error of our computational results over the last four sets BRData, BCData, DPData, and HUData and the relative error of the results obtained by Zribi et al. (2007); Pezzella et al. (2008); and Gao et al.

Table 5 Comparison with algorithms proposed by Zribi et al. (2007), Girish and Jawahar (2009), Pezzella et al. (2008); and Gao et al. (2008) on BRData

Name	LB	hGA	Z.K.K.	dev (%)	G.J.	dev (%)	P.M.C.	dev (%)	G.S.G.	dev (%)
MK01	36	40	41	+2.5	40	0	40	0	40	0
MK02	24	26	28	+7.69	27	+3.85	26	0	26	0
MK03	(204)	204	204	0	204	0	204	0	204	0
MK04	48	61	67	+9.84	62	+1.64	60	−1.64	60	−1.64
MK05	168	173	177	+2.31	178	+2.89	173	0	172	−0.58
MK06	33	62	61	−1.61	78	+25.81	63	+1.61	58	−6.45
MK07	133	141	154	+9.22	147	+4.26	139	−1.42	139	−1.42
MK08	(523)	523	523	0	523	0	523	0	523	0
MK09	299	307	321	+4.56	341	+11.07	311	+1.3	307	0
MK10	165	214	219	+2.34	252	+17.76	212	−0.93	197	−7.94

(2008); with respect to the best-known lower bound. The relative error (*RE*) is defined as:

$$RE = [(MS - LB)/LB] \times 100 \tag{3}$$

where *MS* is the best makespan obtained by the reported algorithm and *LB* is the best-known lower bound. The first column reports the data set, the second column reports the number of instances in each set, the third column reports the average number of alternative machines per operations. The last three columns report the RE of the results obtained by our algorithm; Zribi et al. (2007); Pezzella et al. (2008); and Gao et al. (2008); respectively. The results support the conclusion drawn from Table 3 above that our algorithm is stronger with high degree of flexibility. Also, it shows that our algorithm outperforms the GA proposed by Zribi et al. (2007) and by Pezzella et al. (2008); and produces a comparable quality to the algorithm proposed by Gao et al. (2008). It may worth to mention here that our algorithm has obtained these results with less number of chromosomes than the other two algorithms. For example, Pezzella et al. (2008) used 5,000 chromosomes in all their experiments; and Gao et al. (2008) used on average +2,000 chromosomes, however, our algorithm used on average 700 chromosomes to solve the test cases. Furthermore, the advantage of our approach is that it can work for instances with different

Table 6 Mean relative error over the best-known lower bound

Data set	Num. of ins	Alter.	hGA	Z.K.K.	P.M.C.	G.S.G.
BRData	10	2.59	17.58	21.62	17.53	14.92
BCData	21	1.18	24.74	–	29.56	22.61
DPData	18	2.49	6.83	8.27	7.63	2.12
HUData						
EData	43	1.15	3.92	–	6.00	2.51
RData	43	2	3.68	–	4.42	1.21
VData	43	4.31	0.80	–	2.04	0.09

flexibility. Also, it can be adapted to deal with other criteria than the makespan. This is because the local search algorithm is designed to work in combined neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_3$, rather than only considering neighborhood $\mathcal{N}_4$.

6 Conclusions

In this paper, a hybridized genetic algorithm for the flexible job shop scheduling problem is proposed. The hybrid architecture consists of three main parts, initial population generation heuristic, a local search method, and a genetic algorithm. The experimental results advocate the good performance of the proposed Ini-PopGen heuristic by outperforming some of the existing approaches in the literature. The performance of Ini-PopGen is improved when combining it with the proposed LS. This performance indicates that Ini-PopGen with LS can be used as a stand-alone tool for small to medium sized T-FJSP and P-FJSP. The proposed hGA has the ability to further improve the quality of results obtained by using Ini-PopGen with LS. Thus, the hGA structure is very effective and has a good potential of obtaining optimal or near optimal results. A very strong advantage of the proposed hGA architecture is that the genetic operators of crossover and mutation do not require a repair process to obtain a feasible schedule. Furthermore, the current work maintains the diversity of population by implementing different techniques like the individuals selection method, the mutation techniques, the elitism strategy, and by applying the local search every *d* generations. This allows the GA to explore more solution space whereas Ini-PopGen and LS does the exploitation part.

A future research work would be to extend the proposed hGA to handle more realistic circumstances by including the presence of disturbances and other stochastic measures such as machines breakdown and uncertainties of processing times. Authors are currently studying this direction and initial results are promising.

References

Barnes JW, Chambers JB (1996) Flexible job shop scheduling by tabu search. Graduate Program in Operations Research and Industrial Engineering, The University of Texas, Austin, Technical Report Series, ORP 96-09

Bona B, Brandimarte P, Greco C, Menga G (1990) Hybrid hierarchical scheduling and control systems in manufacturing. IEEE Trans Robot Autom 6(6):673–686

Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41:157–183

Brucker P, Neyer J (1998) Tabu-search for the multi-mode job-shop problem. OR Spektrum 20:21–28

Chan FTS, Chung SH, Chan PLY (2006) Application of genetic algorithms with dominant genes in a distributed scheduling problem in flexible manufacturing systems. Int J Prod Res 44(3):523–543

Chen H, Ihlow J, Lehmann C (1999) A genetic algorithm for flexible job-shop scheduling. Proceedings of the 1999 IEEE international conference on robotics and automation, vol 2. Detroit, Michigan, pp 1120–1125

Cheng R, Gen M, Tsujimura Y (1996) A tutorial survey of job-shop scheduling problems using genetic algorithms–I. Representation. Comput Ind Eng 30(4):983–997

Dauzére-Pérés S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 70:281–306

Falkenauer E, Bouffouix S (1991) A genetic algorithm for job shop. Proceedings of the 1991 IEEE international conference on robotics and automation. Sacramento-California, pp 824–829

French S (1982) Sequencing and scheduling: an introduction to the mathematics of the job-shop. John Wiley, Ellis Horwood Limited, West Sussex

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

Garey MR, Johnson DS, Sethi R (1976) The complexity of flow shop and job-shop scheduling. Math Oper Res 1–2:117–129

Gen M, Cheng R (2000) Genetic algorithms and engineering optimization. Wiley series in engineering design and automation. Wiley, New York

Giovanni LD, Pezzella F (2010) An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. Eur J Oper Res 200:395–408

Girish BS, Jawahar N (2008) Scheduling job shop associated with multiple routings with genetic and ant colony heuristic. Int J Prod Res 99999(2):1–27. doi:10.1080/00207540701824845

Girish BS, Jawahar N (2009) A particle swarm optimization algorithm for flexible job shop scheduling problem. 5th Annual IEEE conference on automation science and engineering. Bangalore, India, 22–25 Aug, pp 298–303

Hart E, Ross P, Corne D (2005) Evolutionary scheduling: a review. Genet Programm Evol Mach 6:191–220

Hmida AB, Haouari M, Huguet M-J, Lopez P (2010) Discrepancy search for the flexible job shop scheduling problem. Comput Oper Res 37:2192–2201

Ho NB, Tay JC (2004) GENACE: an efficient cultural algorithm for solving the flexible job-shop problem. Proceedings of the congress on evolutionary computation CEC2004, pp 1759–1766

Ho NB, Tay JC, EMK Lai (2007) An effective architecture for learning and evolving flexible job-shop schedules. Eur J Oper Res 179:316–333

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

Hussain MF, Joshi SB (1998) A genetic algorithm for job shop scheduling problem with alternate routing. IEEE Int Conf Syst Man Cybern 3:2225–2230

Jia HZ, Nee AYC, Fuh JYH, Zhang YF (2003) A modified genetic algorithm for distributed scheduling problems. J Int Manuf 14:351–362

Kacem I (2003) Genetic algorithm for the flexible job-shop scheduling problem. IEEE Int Conf Syst Man Cybern 4:3464–3469

Kacem I, Hammadi S, Borne P (2002a) Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybern 32–1:1–13

Kacem I, Hammadi S, Borne P (2002b) Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 60(3):245–276

Kumar R, Tiwari MK, Shankar R (2003) Scheduling of flexible manufacturing systems: an ant colony optimization approach. IMechE 217(B):1443–1453

Lee DY, DiCesare F (1994) Scheduling flexible manufacturing systems using petri nets and heuristic search. IEEE Trans Robot Autom 10(2):123–132

Lee EJ, Mirchandani PB (1988) Concurrent routing, sequencing, and setups for a two-machine flexible manufacturing cell. IEEE J Robot Autom 4(3):256–264

Ling QI, Jian-dong Y, Bao LI, Han-cheng YU (2010) Flexible job-shop scheduling problem based on adaptive ant colony algorithm. J Mech Electrical Eng. doi:CNKI:SUN:JDGC.0.2010-02-016

Liu H, Abraham A, Wang Z (2009) A multi-swarm approach to multi-objective flexible job-shop scheduling problems. Fundam Informaticae 95:465–489

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

Mattfeld DC (1996) Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling. Physica-Verlag, Germany, Heidelberg

Mellor P (1966) A review of job shop scheduling. Operat Res Q 17(2):161–171

Mesghouni K, Hammadi S, Borne P (1997) Evolution programs for job-shop scheduling. IEEE international conference on systems, man, and cybernetics, vol 1. Orlando, Florida, pp 720–725

Mirchandani P, Lee EJ, Vasquez A (1988) Concurrent scheduling in flexible automation. Proceedings of the 1988 IEEE international conference on systems, man, and cybernetics, vol 2(8–12), pp 868–872

Murovec B, Šuhel P (2004) A repairing technique for the local search of the job-shop problem. Eur J Oper Res 153:220–238

Najid NM, Dauzére-Pérés S, Zaidat A (2002) A modified simulated annealing method for flexible job shop scheduling problem. IEEE Int Conf Syst Man Cyber 5(6):1–6

Paulli J (1995) A hierarchical approach for the FMS scheduling problem. Eur J Oper Res 86(1):32–42

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

Pinedo M (2002) Scheduling: theory, algorithms and systems. Prentice-Hall, Englewood cliffs, NJ

Tay JC, Wibowo D (2004) An effective chromosome representation for evaluating flexible job shop schedules. Genet Evol Comput 3103:210–221

Wei Q, Qiaoyun L (2009) Solving the flexible job shop scheduling problem based on the adaptive genetic algorithm. Int Forum Comput Sci Technol Appl 1:97–100

White T, Oppacher F (1994) Adaptive crossover using automata. Lecture notes in computer science, Springer Berlin, Heidelberg, vol 866, pp 229–238

Xia W, Wu Z (2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 48(2):409–425

Xing L-N, Chen Y-W, Yang K-W (2009) An efficient search method for multi-objective flexible job shop scheduling problems. J Intell Manuf 20:283–293

Xing L-N, Chen Y-W, Wang P, Zhao Q-S, Xiong J (2010) A knowledge-based ant colony optimization for flexible job shop scheduling problems. Appl Soft Comput 10:888–896

Xu D-S, Ai X-Y, Xing L-N, (2009) An improved ant colony optimization for flexible job shop scheduling problems. Proceedings of the 2009 international joint conference on computational sciences and optimization, vol 1, pp 517–519

Yang J-B (2001) GA-based discrete dynamic programming approach for scheduling in FMS environments. IEEE Trans Syst Man Cybernetics B 31(5):824–835

Zhang GH, Shao XY, Li PG, Gao L (2009) An effective hybrid particle swarm optimization algorithm for multiobjective flexible job-shop scheduling problems. Comput Ind Eng 56(4):1309–1318

Zribi N, Kacem I, Kamel AE (2007) Assignment and scheduling in flexible job-shops by hierarchical optimization. IEEE Trans Syst Man Cybernetics C Appl Rev 37(4):652–661

Author Biographies

Nasr Al-Hinai is a Ph.D. candidate in the department of Mechanical and Industrial Engineering, University of Manitoba. He received his M.Sc. from the department of Mechanical, Manufacturing, and Aerospace Engineering, UMIST, UK in 2003. His research interest is in the area of scheduling optimization, and operations research.

T. Y. ElMekkawy is an associate professor at the department of Mechanical and Industrial Engineering, University of Manitoba. He received his B.Sc. and an M.Sc. from the Department of Mechanical Design and Production, Cairo University, Egypt in 1990 and 1994, respectively. He received his Ph.D. from the Department of Industrial and Manufacturing Engineering, University of Windsor, Canada, in 2001. He has joined the University of Manitoba (UM) in 2003 after working for 2 years in automotive industry, Ontario, Canada. His research interest is in the areas of scheduling optimization, and applications of operational research and industrial engineering techniques to improve healthcare delivery. He has published many papers in international journals, such as *IJPR*, *IJPE*, *IJCIM*, *IJAMT*, *IJOR,* and *CIRP Annals*.