

Solving Multiple-Objective Flexible Job Shop Problems by Evolution and Local Search

Nhu Binh Ho and Joc Cing Tay, *Senior Member, IEEE*

Abstract—Finding realistic schedules for flexible job shop problems has attracted many researchers recently due to its nondeterministic polynomial time (NP) hardness. In this paper, we present an efficient approach for solving the multiple-objective flexible job shop by combining evolutionary algorithm and guided local search (GLS). Instead of applying random local search to find neighboring solutions, we introduce a GLS procedure to accelerate the process of convergence to Pareto-optimal solutions. The main improvement of this combination is to help diversify the population toward the Pareto front. A branch and bound algorithm for finding the lower bounds of multiple-objective solutions is also proposed. Experimental results indicate that the multiple-objective Pareto-optimal solutions of our algorithms dominate previous designs for solving the same benchmarks while incurring less computational time.

Index Terms—Evolutionary algorithm (EA), guided local search (GLS), multi-objective flexible job shop problems.

I. INTRODUCTION

MANY real-world scheduling problems involve simultaneous optimization of a set of conflicting multiple objectives. In particular, the scheduling task for manufacturing is concerned with assigning n jobs to m machines so as to minimize some conflicting objective functions. When considering multiple objectives, there may not exist a unique solution that is the best for all objectives (global minimum or maximum). Alternatively, a set of solutions that are superior to the rest of solutions in the search space are the targets to achieve. A popular model that has been well studied is the job-shop scheduling problem (JSP). It is one of the hardest scheduling problems to solve for optimality [1] and is nondeterministic polynomial time (NP) hard [2]. In the JSP, the route of every job is fixed and every operation of a job is allocated a unique machine for processing. However, the limitation of allocating only one job to one machine can lead to a bottleneck on the most busy machines on the shop floor. In practice, the machine environment is also more complex. These busy machines are duplicated to balance their overall workload and to reduce the flow time of the jobs [3]. This variation is known as the flexible JSP (FJSP). It extends the definition of the JSP by allowing an operation to be processed without interruption on one of a set of predefined machines. Mati and Xie [4] considered the general FJSP where each operation can be processed by any machine from a given set associated

with the operation, and its processing time would depend on the selected machine. They proved that with two machines for minimizing *makespan* (the maximum completion time of all operations), the FJSP is NP-hard. In addition, the FJSP with multiple objectives is also NP-hard [4]. Many techniques have been considered for solving the FJSP. Enumerative methods such as branch and bound (B&B) [5] can guarantee optimal solutions, but are computationally expensive even for small-sized problems. Other approximation and randomized algorithm methods such as evolutionary algorithms (EAs) [6]–[10], simulated annealing [11], or tabu search (TS) [12] have demonstrated an efficacy for solving FJSPs.

The combination of EAs and local search methods (known as *memetic algorithms*) for solving scheduling problems has been increasingly studied by researchers and good solutions [13], [14] have been obtained. The local search methods by themselves have also achieved promising results [11], [12]. These approaches combine the explorative search of EA and the exploitative search of local search together to find better solutions. However, issues of time and space complexities need to be addressed. In order to reduce these complexities, the application of intelligent heuristics to select promising solutions when applying local search will need further investigation. Considering both advantages and limitations of previous approaches, we propose a multi-objective EA with guided local search (MOEA-GLS) algorithm to solve multi-objective FJSPs. The algorithm comprises three parts: MOEA, GLS, and elitism memory. Instead of simply using a randomized local search for finding neighbors of a schedule, the local search part in every generation is intelligently guided by an efficient selection toward promising areas. These mechanisms guarantee that only the moves that obtain better solutions are visited. Therefore, more good-quality solutions can be obtained and the computational time is also reduced. In this paper, three objectives are considered: *minimization of makespan, critical machine workload, and total workload of all machines*. We validate the efficacy of the MOEA-GLS algorithm to solve common benchmark problems from [6], [7], and [11]. The experimental results indicate that the multi-objective results of our algorithm dominate the other approaches for solving the same benchmarks in shorter computational time.

The paper is organized as follows. Section II gives the formal definition of the FJSP. Section III reviews recent related works for solving the FJSP and classifies current approaches for solving multi-objective optimization problems using EAs. Section IV introduces lower bounds for each single or individual objective in multi-objective FJSPs. The B&B algorithm for finding multiset values of multi-objective FJSPs is also presented. Section V describes GLS and the integration of MOEA

Manuscript received January 15, 2007; revised September 26, 2007 and January 15, 2008. This work was supported in part by Nanyang Technological University and in part by CEI Contract Manufacturing Limited Company, Singapore. This paper was recommended by Associate Editor J. Lazansky.

The authors are with the Evolutionary and Complex Systems Program, School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore (e-mail: honb@pmail.ntu.edu.sg; asjctay@ntu.edu.sg).

Digital Object Identifier 10.1109/TSMCC.2008.923888

and GLS. It also provides some theoretical considerations and the procedure to find the best moves. Section VI analyzes the performance results of MOEA-GLS when applied to solve common benchmarks in literature. Finally, Section VII gives some concluding remarks and directions for future work.

II. PROBLEM DEFINITION

The multi-objective FJSP with availability constraints and functionally related machines is formulated as follows.

- 1) Let $J = \{J_i\}_{1 \leq i \leq n}$, indexed i be a set of n jobs to be scheduled.
- 2) Each job J_i consists of a predetermined sequence of operations. Let $O_{i,j}$ be operation j of J_i .
- 3) Let $M = \{M_k\}_{1 \leq k \leq m}$, indexed k be a set of m machines.
- 4) Each machine can process only one operation at a time.
- 5) Each operation $O_{i,j}$ is processed without interruption on M_k in a given set $\mu_{i,j} \subset M$ with $p_{i,j,k}$ time units.

Let r_i and C_i be the release date and the completion date of job J_i . W_k is the workload of machine M_k . It is the summation of processing times of operations that are processed on machine M_k . Three objectives (which have been used to evaluate the efficacy of other algorithms in solving multi-objective FJSPs [6], [7], [11]) are used in this paper, namely:

- 1) minimization of overall completion time (*makespan*): $F_1 = \max \{C_i \mid i = 1, \dots, n\}$;
- 2) minimization of critical machine workload: $F_2 = \max \{W_k \mid k = 1, \dots, m\}$;
- 3) minimization of total workload of all machines: $F_3 = \sum p_{i,j,k}$.

The task is to find a set of solutions that are superior among all the solutions when all objectives are considered. They are known as *Pareto-optimal* solutions [15].

The FJSP can also be considered to be a multipurpose machine (MPM) job-shop [16]. Using the $\alpha|\beta|\gamma$ notation of Graham *et al.* [17], the problem that we wish to solve can be denoted by:

$$J \text{ MPM } | \text{prec } r_j | F_1 F_2 F_3$$

where J denotes a job-shop problem, MPM denotes a multi-purpose machine, $prec$ represents a set of independent *chains*, while r_j represents the *release date* given to each job, F_1 (or C_{\max}) represents *makespan*, F_2 represents the critical machine workload, and F_3 represents the total workload of all machines.

In this paper, we will assume the following:

- 1) all machines are available at time 0;
- 2) each job has its own release date;
- 3) the order of operations for each job is predefined and invariant;
- 4) the machine can execute only one operation at a time.

III. LITERATURE REVIEW

EAs have been used widely to solve multi-objective optimization problems. Generally, they can be classified into three approaches: population based, aggregation function, and Pareto based [18].

Population-based approaches, such as VEGA [19], are based on a division of the current population into s sub-populations, where s is the number of objectives. At each generation, s sub-populations are generated by performing proportional selection according to each objective before being shuffled and recombined into a single population. The crossover and mutation operators are then applied as usual to this new population. The drawback of this approach is the focus on one objective per sub-population at a time. Therefore, the results that are good for more than one objective may be discarded before recombining together to form a new population.

The aggregation function approaches combine all the objectives of the optimization problems into a single function [7], [13]. An example of an aggregation function is $\min \sum_{i=1}^k w_i F_i$, where w_i is the weight of a single objective F_i . This approach is straightforward to apply to any multi-objective optimization problem. However, due to the difficulty of setting the weight vector's values for exploiting the desired area, it is not sufficient for solving non-convex objective spaces [18].

Pareto-based approaches [18] use the concept of domination to find optimal results. A solution X dominates a solution Y if the solution X has at least one objective that fares better than the corresponding objective in solution Y , all others being equal. The family of all non-dominated alternative solutions is denoted as the Pareto-optimal set, Pareto set for short or Pareto-optimal front. There are many Pareto-based approaches that have been developed so far in the literature. The most popular methods are NSGA-II [20] and SPEA [21] that depend on elitist selection, fitness sharing, and Pareto ranking. Elitist selection preserves the elite individuals from the last generation, fitness sharing degrades the fitness values of all competing members of a niche as the niche size increases, while Pareto ranking uses dominance concepts for selection to eliminate inferior individuals. The challenge of Pareto-based approaches is to keep the diversity of the population toward the Pareto front of the problems.

Recently, many researchers have studied the combination of EA and local search for solving multi-objective scheduling problems (also known as memetic algorithms [22]–[24]). Some studies also apply memetic algorithms to solve flowshop and timetabling problems [13], [14]. Ishibuchi *et al.* [13] combines EA and local search for solving the multi-objective flow shop. The weight vectors are generated randomly in each generation to combine all objectives. At the end of each generation, all schedules in the population are improved by local search after a predefined number of steps. Kacem *et al.* [7] apply fuzzy logic to control the way to find weight vectors for solving multi-objective FJSPs, while Xia and Wu [11] uses simulated annealing integrated with local search. In the latter approach, two randomly selected nearby operations on each machine are swapped in the local search. Similar to other aggregation function approaches, the algorithms described earlier also face issues of finding suitable weight vectors to diversify the results toward the Pareto front. This is computationally expensive when all solutions are used to evaluate new weight vectors. Furthermore, if random local search algorithms are applied, they do not guarantee improvement in the obtained results. The difference between this research and previous approaches on solving multi-objective

FJSPs is that we use a mechanism of GLS to improve *only* the best solutions. Furthermore, we have adopted the Pareto-based approaches for ranking the solutions in each generation. We will demonstrate the efficacy of the MOEA-GLS algorithm in comparison to other approaches by using the benchmarks in [7].

IV. LOWER BOUNDS FOR THE MULTI-OBJECTIVE FJSP

In this section, we develop a set of lower bounds that will be used to analyze the multi-objective FJSPs results obtained by our algorithms. The lower bounds for single objective, e.g., *makespan*, has been investigated widely in literature for solving classical job-shop problems, especially in B&B algorithms [25], [26]. For parallel machine problems, lower bounds are investigated in [27]. For FJSPs, to the best of our current knowledge, only Kacem *et al.* [7] has introduced lower bounds for multi-objective FJSPs. However, the limitation of their approach is the local consideration upon each objective. Each lower bound value for each objective is evaluated in isolation, without consideration of other objectives. In multi-objective optimization, it is well known that there is often a tradeoff between objectives (see Section III). Extremization of one objective can lead to a change in other objective values. In this section, we propose two approaches to calculate the lower bounds for multi-objective FJSPs described in Section II. First, we introduce *single* lower bound values that have been derived from the results of Kacem *et al.* [7]. Second, we propose a B&B algorithm to find the *multi-lower* bound set for multi-objective FJSPs. The multi-lower bound set is stronger than the single lower bound values since a set of multi-objective lower bounds is explored at the same time on a reduced search space during B&B. With this set of lower bounds, we are able to evaluate how far the current solution is to the *Pareto front*.

Before we discuss theoretical works and the B&B algorithm for finding multi-lower bound sets for multi-objective FJSPs, we introduce notations and definitions that will be used.

- Let n_i denote the number of operations of job J_i
- $U = \{O_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq n_i\}$ be the set of all operations;
- $N = \sum_{i=1}^n n_i = |U|$ be the total number of operations;
- N_k be the predefined maximum number of operations that can be processed on machine M_k ;
- U_{M_k} be the set of possible operations (of size N_k) that are processed on machine M_k ;
- $\delta_{i,j} = \min_k \{p_{i,j,k}\}$, $1 \leq k \leq m$ denote the smallest processing time of $O_{i,j}$ among a set of m machines;
- $\text{est}(O_{i,j})$ denote the earliest start time of $O_{i,j}$;
- $\text{ect}(O_{i,j})$ denote the earliest completion time of $O_{i,j}$;
- $\text{srpt}(O_{i,j})$ denote the smallest remaining processing time of job J_i after processing $O_{i,j}$;
- $\lceil x \rceil$ denotes the smallest integer that is equal or larger than x (or ceiling of x).

Definition 1: Pareto Dominance (for Minimization): A vector $u = (u_1, u_2, \dots, u_n)$ is said to dominate vector $v = (v_1, v_2, \dots, v_n)$ (denoted by $u \preceq v$) if and only if u is partially less than v , $\forall i \in \{1, 2, \dots, n\}$ $u_i \leq v_i \wedge \exists k \in \{1, 2, \dots, n\} : u_k < v_k$.

Lemma 1:

$$\text{est}(O_{i,j}) = \begin{cases} r_i, & \text{if } j = 1 \\ \sum_{k=1}^{j-1} \delta_{i,k}, & \text{otherwise.} \end{cases}$$

Proof: Obvious, earliest start time of one operation is equal to sum of minimum processing time of its previous operations.

Lemma 2:

$$\text{ect}(O_{i,j}) = \text{est}(O_{i,j}) + \delta_{i,j}.$$

Proof: Obvious, earliest completion time of operation $O_{i,j}$ is equal to its earliest start time plus the smallest processing time of $O_{i,j}$.

Lemma 3:

$$\text{srpt}(O_{i,j}) = \begin{cases} 0, & \text{if } j = n_i \\ \sum_{k=j+1}^{n_i} \delta_{i,k}, & \text{otherwise.} \end{cases}$$

Proof: Obvious, if j is equal to n_i , then $O_{i,j}$ is the last operation of job J_i . Therefore, the smallest remaining processing time of J_i after processing $O_{i,j}$ is 0. Otherwise, it is the sum of smallest processing times $\delta_{i,k}$ of the remaining operations of job J_i .

Lemma 4: If $O_{i,j}$ is an immediate predecessor of operation $O_{k,l}$ on machine M_m such that $\text{est}(O_{i,j}) \leq \text{est}(O_{k,l})$, then the idle time between $O_{i,j}$ and $O_{k,l}$ is

$$\Delta(O_{i,j}, O_{k,l}) = \max\{0, \text{est}(O_{k,l}) - \text{ect}(O_{i,j})\}.$$

Proof: If $\text{est}(O_{k,l}) \leq \text{ect}(O_{i,j})$, then $O_{k,l}$ has to wait until $O_{i,j}$ is finished on M_m . Therefore, the idle time between $O_{i,j}$ and $O_{k,l}$ on M_m is 0. Otherwise, the idle time is equal to $\text{est}(O_{k,l}) - \text{ect}(O_{i,j})$. This lemma is justified.

A. Single Lower Bound Values for Multi-objective FJSPs

1) Lower Bound for Total Workload:

Theorem 1:

$$F_3^* \geq \left(\sum_{i=1}^n \sum_{j=1}^{n_i} \delta_{i,j} \right)$$

Proof: Obvious, the total workload cannot exceed the sum of smallest processing times of each operation. Therefore, the theorem is justified.

2) Lower Bound for the Critical Machine Workload:

Theorem 2:

$$\text{LB} \geq \left\lceil \frac{\sum_{i=1}^n \sum_{j=1}^{n_i} \delta_{i,j}}{m} \right\rceil.$$

Proof: Obvious, at least one machine has processing time that is greater or equal to the average smallest processing time of all machines. Since the critical machine workload value is an integer, the lower bound is at least the ceiling value of the average minimal processing times of all machines. Therefore, the theorem is justified.

The average number of operations that are processed one machine is

$$\bar{N} = \left\lceil \frac{N}{m} \right\rceil.$$

Therefore, there exists at least one machine so that its number of operations is equal or larger than \bar{N} . We adopt the idea from Kacem *et al.* [7] where the set of operations U is divided into two subsets: a subset $P_{\bar{N}_k}$ where \bar{N} operations are processed on M_k and remaining operations $U - P_{\bar{N}_k}$ that are processed on the remaining $m - 1$ machines.

Definition 2: Let V be the set of machines where the maximum number of operations of each machine in V is at least \bar{N}

$$V = \{M_i : N_i \geq \bar{N}\}.$$

Therefore, $\forall M_k \in V$, the number of possible operation combinations serviced by M_k is given by $P_{\bar{N}_k} = \binom{N_k}{\bar{N}}$. Let the set P be the set of all $P_{\bar{N}_k}$ subsets for all machines $M_k \in V$. For each set $P_{\bar{N}_k}$:

- 1) The workload of M_k on processing \bar{N} operations:

$$\beta_{1,k} = \sum_{l=1}^{\bar{N}} p_{i_l, j_l, k}, \quad \text{where } O_{i,j} \in P_{\bar{N}_k}.$$

- 2) The average workload of $N - \bar{N}$ remaining operations in $(m - 1)$ remaining machines is

$$\beta_{2,k} = \left\lceil \frac{\sum_{i,j} \delta_{i,j}}{m-1} \right\rceil, \quad \text{where } O_{i,j} \in \{U - P_{\bar{N}_k}\}.$$

Theorem 3: When considering all subsets of set P , the lower bound for total machine workload is calculated as follows:

$$\text{LB} \geq \min_{P_{\bar{N}_k} \in P} \{\max(\beta_{1,k}, \beta_{2,k})\}.$$

Proof: We consider a single set $P_{\bar{N}_k} \in P$. By definition of $\beta_{1,k}$ and $\beta_{2,k}$, the lower bound is $\max(\beta_{1,k}, \beta_{2,k})$. Furthermore, at least one machine satisfies the condition that its number of operations is equal or larger than \bar{N} . Therefore, the theorem is justified.

The time complexity of Theorem 3 is exponential since $\binom{N_k}{\bar{N}}$ selections have to be checked. The lower bound obtained by Theorem 4 to be described as follows is weaker than Theorem 3, but its time complexity is polynomial.

From Definition 2, $\forall M_k \in V$, let Q_{1,M_k} be a subset of U_{M_k} where \bar{N} operations with smallest values of $p_{i,j,k}$ are selected, let Q_{2,M_k} be a subset of U_{M_k} where \bar{N} operations with largest values of $\delta_{i,j}$ are selected. Then

- 1) The smallest workload in considering \bar{N} operations on M_k is

$$\lambda_{1,k} = \sum_{l=1}^{\bar{N}} p_{i_l, j_l, k}, \quad \text{where } O_{i,j} \in Q_{1,M_k}.$$

- 2) The smallest average workload of $N - \bar{N}$ remaining operations in $(m - 1)$ remaining machines is

$$\lambda_{2,k} = \left\lceil \frac{\sum_{i,j} \delta_{i,j}}{m-1} \right\rceil, \quad \text{where } O_{i,j} \in \{U - Q_{2,M_k}\}.$$

Theorem 4: When considering all machines belonging to V , the lower bound for total machine workload is calculated as follows:

$$\text{LB} \geq \min_{M_k \in V} (\max(\lambda_{1,k}, \lambda_{2,k})).$$

Proof: The operations in Q_{1,M_k} are the \bar{N} operations that have smallest $p_{i,j,k}$ values in U_{M_k} . Therefore, $\lambda_{1,k}$ is the smallest sum of $p_{i,j,k}$ of \bar{N} operations in U_{M_k} . Similarly, the sum of $\delta_{i,j}$ of \bar{N} operations in Q_{2,M_k} is the largest value among \bar{N} operations in U_{M_k} . Therefore, $\lambda_{2,k}$ is the smallest workload of $N - \bar{N}$ remaining operations in $(m - 1)$ remaining machines. For each machine $M_k \in V$, the lower bound of critical workload is at least $\max(\lambda_{1,k}, \lambda_{2,k})$. The theorem is justified.

From Theorems 2 and 4, it follows that the lower bound of the critical machine workload is

$$F_2^* \geq \max \left(\left\lceil \frac{\sum_{i=1}^n \sum_{j=1}^{n_i} \delta_{i,j}}{m} \right\rceil, \min_{M_k \in V} (\max(\lambda_{1,k}, \lambda_{2,k})) \right).$$

- 3) Lower Bound for Makespan:

Theorem 5:

$$\text{LB} \geq \max_{1 \leq i \leq n} \left(r_i + \sum_{j=1}^{n_i} \delta_{i,j} \right).$$

Proof: Based on precedent constraints of operations for each job, the result is justified.

Theorem 2 gives the lower bound of the critical machine workload by dividing the total smallest processing times by the number of machines. The sum of the earliest start time of each operation and the machine's critical workload gives the lower bound for *makespan*. Lemma 5 described later generalizes the result of [7] for the case where the number of operations is smaller than the number of machines.

Lemma 5: Given a tuple W of z operations that are ordered in non-decreasing earliest start time:

$$W = \langle O_{i_1, j_1}, O_{i_2, j_2}, \dots, O_{i_z, j_z} \rangle$$

such as $\text{est}(O_{i_1, j_1}) \leq \text{est}(O_{i_2, j_2}) \leq \dots \leq \text{est}(O_{i_z, j_z})$.

The lower bound for the average idle time of r machines that process these z operations is

$$I(r, W) = \begin{cases} \frac{\sum_{k=1}^r \text{est}(O_{i_k, j_k})}{r}, & \text{if } z \geq r \\ \frac{\sum_{k=1}^z \text{est}(O_{i_k, j_k})}{z}, & \text{otherwise.} \end{cases}$$

Proof: In order to minimize the *makespan*, we assume that all r machines are used to process the operations in W . In the case where $z < r$, only z machines are used, else (when $z \geq r$) all r machines are applied. For the first machine, the idle time is

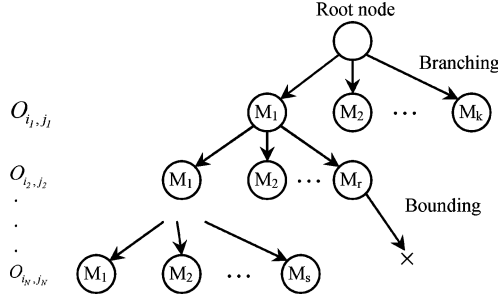


Fig. 1. B&B tree for finding the multi-lower bound set.

$\text{est}(O_{i_1, j_1})$. For the second machine, the idle time is $\text{est}(O_{i_2, j_2})$, and so on. Therefore, the lower bound of average idle time of r machines to process z operations in W is $I(r, W)$.

Theorem 6: The lower bound of makespan is

$$\text{LB} \geq \left[I(m, U) + \frac{\sum_{i=1}^n \sum_{j=1}^{n_i} \delta_{i,j}}{m} \right].$$

Proof: From Lemma 5, the lower bound of the average idle time of m machines on U operations is $I(m, U)$. In addition, at least one machine has a processing time that is at least the average processing time for all m machines. The lower bound of the makespan is equal to the sum of the average idle time and the average processing time of m machines. Therefore, the theorem is justified.

From Theorems 5 and 6, it follows that the lower bound of the *makespan* for a FJSP is

$$F_1^* = \max \left(\max_{1 \leq i \leq n} \left(r_i + \sum_{j=1}^{n_i} \delta_{i,j} \right), \left[I(m, U) + \frac{\sum_{i=1}^n \sum_{j=1}^{n_i} \delta_{i,j}}{m} \right] \right).$$

B. Multi-lower Bound Set for Multi-objective FJSPs

The B&B for finding the multi-lower bound set is performed by representing all feasible lower bounds for multiple objectives in each subtree of the search tree through an assignment of each operation to an available machine that can process it. The underlying backtracking mechanism represents each variable as a machine, and each variable domain as those operations that can be processed by the machine. The constraints are the precedence of the assigned operations.

For instance, if O_{i_1, j_1} is the first operation that can be processed on each machine in the set $\mu_{i_1, j_1} = \{M_1, M_2, \dots, M_k\}$. We assign O_{i_1, j_1} to machine M_1 in μ_{i_1, j_1} followed by assigning O_{i_2, j_2} to a machine in $\mu_{i_2, j_2} = \{M_1, M_2, \dots, M_r\}$ until the assignment of the last operation O_{i_N, j_N} is reached while satisfying all constraints (see Fig. 1).

Consider the total FJSP with N operations and m machines, where each operation can be processed on any of m machines. The maximum number of nodes to traverse by B&B would be in $O(m^N)$. In order to reduce this search complexity, upper bounds are generated from heuristic methods and lower bounds are ob-

tained by the theorems from Section IV-A. The B&B algorithm to find the multi-lower bound set for an FJSP P is as follows.

Step 1) Using heuristic methods (e.g., dispatching rules) for solving P , the obtained result is a set of feasible schedules with a set of multiple-objective values S . Let S be the initial candidate set of upper bound objective values.

Step 2) Using the theorems in Section IV-A to evaluate the lower bounds for single objectives, we set

$$F_1 = F_1^* \quad F_2 = F_2^* \quad F_3 = F_3^*.$$

Step 3) Order all N operations of P in non-decreasing earliest start time and assign them to list L , such that

$$L = \{O_{i_1, j_1} \quad O_{i_2, j_2} \quad O_{i_3, j_3} \quad \dots \quad O_{i_N, j_N}\}.$$

$$\begin{matrix} 1 & 2 & 3 & \dots & N \end{matrix}$$

Set $l = 0$.

Step 4) Get operation $O_{x,y}$ at the next position $(l + 1)$. The lower bounds before assigning $O_{x,y}$ to a machine in $\mu_{x,y}$ is (F_1, F_2, F_3) .

Step 5) **Branching:** for each machine M_k in $\mu_{x,y}$:

Assign $O_{x,y}$ to be processed.

Calculate the current workload:

$$A = \text{current workload of } M_k + p_{i,j,k}.$$

Calculate stop time:

$$B = \max(\text{stop time of } M_k + p_{i,j,k}, \text{ect}(O_{x,y})).$$

Calculate the earliest finishing time of job J_x :

$$C = B + \text{srpt}(O_{x,y}).$$

Step 6) Compute the lower bound of the three objectives for this subtree in the following order:

a) Total workload: $F_3 = F_3 + (p_{x,y,k} - \delta_{x,y})$.

b) Critical machine workload:

$$F_2 = \max \left(F_2, \left\lceil \frac{F_3}{m} \right\rceil, A \right).$$

c) **Makespan:** $F_1 = \max(F_1, F_2, B, C)$.

Step 7) **Bounding:** if (F_1, F_2, F_3) is dominated by any solution in S , go to Step 9.

Step 8) If $O_{x,y}$ is not the last operation in list L ($l < n$), go to Step 4. Otherwise, update list S by removing all solutions that are Pareto dominated by (F_1, F_2, F_3) , then add (F_1, F_2, F_3) to S .

Step 9) Stop.

At Step 1, the upper bounds are generated by heuristic algorithms for solving FJSPs. If these solutions are nearby Pareto-optimal solutions, the number of traveling nodes will be reduced significantly. The lower bounds for three single objectives are evaluated at Step 2. Step 5 is the branching step where an operation $O_{x,y}$ is assigned to a machine in $\mu_{x,y}$. Instead of waiting for the assignment of the last operation O_{i_N, j_N} to calculate each objective value, we look ahead to evaluate the objectives of the current subtree. Therefore, the worst-case results can be ignored early to reduce computational time. The total workload

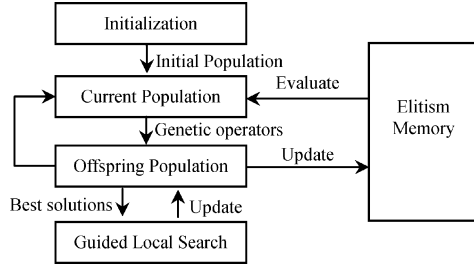


Fig. 2. Overall control flow of the proposed MOEA-GLS algorithm.

in Step 6(a) is estimated by increasing the current processing time of $O_{i,j}$ on machine M_k by an amount relative to its smallest possible processing time: $(p_{x,y,k} - \delta_{x,y})$. The critical machine workload is the maximum value of the previous critical machine workload F_2 , the average processing time of one machine $\lceil \frac{F_3}{m} \rceil$ and A , the current workload of machine M_k . In order to estimate the *makespan*, two more values: B , stop time of machine M_k and C , the earliest finishing time of job J_x are estimated. At the root node, all stop times of machines are set to 0. If the earliest start time of $O_{x,y}$ is smaller than the current stop time of M_k before assigning $O_{x,y}$, B is set to the stop time of $M_k + p_{i,j,k}$. Otherwise, B is set to the earliest completion time of $O_{x,y}$: $\text{ect}(O_{x,y})$. C is estimated by B , the stop time of $O_{x,y}$ on machine M_k , and the shortest remaining processing time of job J_x from $O_{x,y}$. Therefore, the new *makespan* is the maximum value of the *makespan*, the critical machine workload, the stop time of machine M_k , and the earliest finishing time of job J_x . The obtained results of the B&B algorithm are the set s that contains the Pareto-optimal lower bound values for a multi-objective FJSP. They can be used to evaluate the efficacy of an algorithm for solving a multi-objective FJSP.

V. MOEA-GLS ALGORITHM

The overall control flow of the MOEA-GLS algorithm is summarized in Fig. 2. To generate a good and diversified initial population, we employ composite dispatching rules [8] for constructing the initial population. These rules perform two tasks simultaneously: balancing the workload of each machine and ordering the operations in the waiting list of each machine so that good solutions can be obtained. The GLS module uses the procedure described in Section IV-A. Coello [18] mentions that the second generation of MOEAs have introduced the use of an elitism selection strategy. Similar to previous research in MOEA [13], [21], [28], we adopt an elitism memory to keep all non-dominated solutions that have been found after each generation. This elitism memory is then used to update the current population to enhance its quality of solutions.

Section V-A gives common definitions that will be used. Theoretical considerations for GLS in Fig. 2 are also discussed. Section V-B then presents detail description of our MOEA-GLS algorithm and the selection of designed parameters to reduce computational time.

A. Finding Neighbors Using Guided Local Search

For a given schedule of the FJSP, a neighbor can be obtained by moving an operation from one machine to another

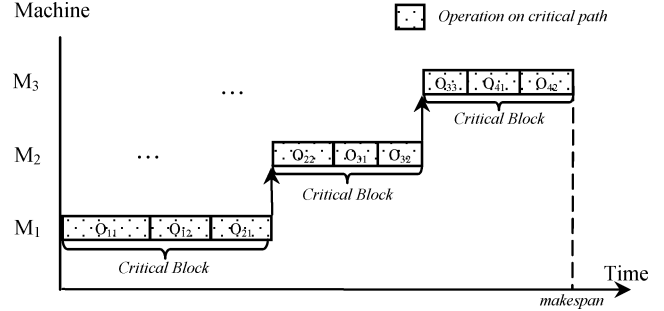


Fig. 3. Example of critical path and critical blocks.

machine. However, not all moves improve the current results and a complete enumeration of the neighbors will be expensive. The challenge is how to find the best move to improve multiple objectives simultaneously. Before presenting theorems that are used to guide the local search, some definitions are necessary.

Definition 3: The *critical path* of a schedule is a list of operations whose total process time is the minimum time that all jobs take to complete. One schedule can have more than one critical path.

Definition 4: A *critical block* is a set of consecutive operations on the critical path processed by the same machine. One machine can be associated with more than one critical block on a given critical path.

Fig. 3 gives an example of a critical path and critical blocks.

Definition 5: The machine M_k in a schedule is termed the *most loaded machine* if its workload (WL) is equal to the critical machine workload (MW), which we denote as a function, $WL(M_k) = MW$. One schedule can have more than one most loaded machine.

Definition 6: The *precedence earliest start time* of an operation $[pest(O_{i,j})]$ is equal to the stop time of its immediate precedence operator $O_{i,j-1}$. If it is the first operation, $pest(O_{i,j})$ is equal to the release date of the corresponding job.

Theorem 7 (minimizing the total workload): When moving an operation $O_{i,j}$ from machine M_k to another machine $M_{k'}$, the new total workload is minimal (i.e., at most the previous total workload) if the processing time of $O_{i,j}$ on machine $M_{k'}$ is at most the processing time of $O_{i,j}$ on machine M_k .

Proof: Let $p_{i,j,k}$ be processing time of $O_{i,j}$ on machine M_k , the total workload of all machines is

$$TW = \sum_{i=1}^n \sum_{j=1}^{n_i} p_{i,j,k} = p_{1,1,l} + \dots + p_{i,j,k} + \dots + p_{n,p,q}$$

where n is number of jobs, n_i is number of operations of job i and k is the index of the machine that processes operation $O_{i,j}$.

When moving $O_{i,j}$ from machine M_k to machine $M_{k'}$, the new processing time of this operation on machine $M_{k'}$ is $p_{i,j,k'}$. The new total workload of all machines is

$$TW' = \sum_{i=1}^n \sum_{j=1}^{n_i} p_{i,j,k} = p_{1,1,l} + \dots + p_{i,j,k'} + \dots + p_{n,p,q}$$

Since $p_{i,j,k'} \leq p_{i,j,k}$, we obtain $TW' \leq TW$. Therefore, the total workload is minimal.

Theorem 8 (Minimizing the Critical Machine Workload):

When moving an operation $O_{i,j}$ from one of the most loaded machines (e.g., M_k) to another machine $M_{k'}$ on a schedule, the critical machine workload is minimal (i.e., at most the critical machine workload MW) if the workload on machine $M_{k'}$ plus the processing time of $O_{i,j}$ on machine $M_{k'}$ is at most MW.

Proof: Let M_k be one of the most loaded machines, then by Definition 5

$$MW = WL(M_k) = \sum_{M_k} p_{i,j,k}$$

where $p_{i,j,k}$ is processing time of $O_{i,j}$ on machine M_k . Similarly, the workload of machine $M_{k'}$ is

$$WL(M_{k'}) = \sum_{M_{k'}} p'_{i,j,k'}$$

When moving an operation $O_{i,j}$ from machine M_k (with processing time $p_{i,j,k}$) to machine $M_{k'}$ (with processing time $p_{i,j,k'}$), the workload on machine M_k decreases to

$$WL'(M_k) = WL(M_k) - p_{i,j,k}$$

while the workload on machine $M_{k'}$ increases to

$$WL'(M_{k'}) = WL(M_{k'}) + p_{i,j,k'}$$

If $WL'(M_{k'}) \leq WL(M_k)$ or $WL(M_{k'}) + p_{i,j,k'} \leq MW$, then the new critical machine workload is minimal. Otherwise, the new critical machine workload (that is equal to the workload on machine $M_{k'}$) will be larger than MW.

Theorem 9 (Makespan Invariance): The makespan of a schedule is invariant when moving operations that are not on *critical paths* to other machines.

Proof: The makespan is the total processing time of all operations of a given critical path. If moving any operation that does not belong to the critical path to another machine (not on this path), the current makespan remains unchanged. Therefore, the makespan cannot be reduced by such an action.

Theorem 10 (Minimizing the Makespan): When moving an operation $O_{i,j}$ belonging to machine M_k (with processing time $p_{i,j,k}$) on a *critical path CP* to machine $M_{k'}$ (with processing time $p_{i,j,k'}$). If its immediate precedence and successor operations are also on the same *critical path CP*, the makespan cannot be reduced if $p_{i,j,k'}$ is equal or larger than $p_{i,j,k}$.

Proof: Let C_{\max} be the makespan of the schedule before moving $O_{i,j}$ from machine M_k to machine $M_{k'}$. Let $O_{i,j-1}$ and $O_{i,j+1}$ be precedence and successor operations of $O_{i,j}$, respectively. $O_{i,j}$ cannot start before the stop time of $O_{i,j-1}$. Similarly, $O_{i,j+1}$ cannot start before the stop time of $O_{i,j}$. Since $O_{i,j-1}$ and $O_{i,j+1}$ are both on the *critical path CP*, the lower bound of the new makespan, C'_{\max} is

$$C'_{\max} = (C_{\max} - p_{i,j,k}) + p_{i,j,k'}$$

Since $p_{i,j,k'} \geq p_{i,j,k}$, $C'_{\max} \geq C_{\max}$, the makespan cannot be reduced.

When moving an operation $O_{i,j}$ on machine M_k (with processing time $p_{i,j,k}$) of a schedule to new machine $M_{k'}$ (with processing time $p_{i,j,k'}$), the operation $O_{i,j}$ can only be processed after the completion of its immediate precedence opera-

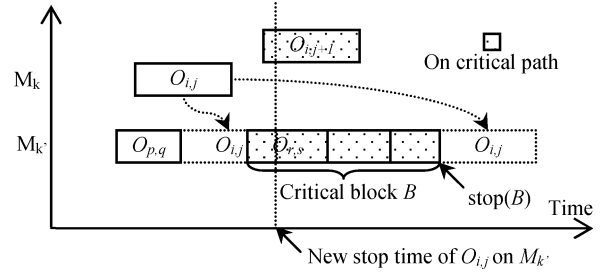


Fig. 4. Finding a suitable time to process $O_{i,j}$ on machine $M_{k'}$.

tion $O_{i,j-1}$ [$\text{pest}(O_{i,j})$]. Using the constraint of $\text{pest}(O_{i,j})$, we identify the operation $O_{p,q}$ in machine $M_{k'}$, which has a stop time that is smaller or equal to $\text{pest}(O_{i,j})$, $O_{i,j}$ can then be processed after $O_{p,q}$. However, it may increase the current value of makespan if both $O_{i,j+1}$ and the immediate next operation $O_{r,s}$ of $O_{p,q}$ on machine $M_{k'}$ are on critical paths. This situation is presented in Fig. 4 later.

In Fig. 4, both the immediate successor operation $O_{i,j+1}$ and $O_{r,s}$ are on critical paths. Let set B be the critical block that contains $O_{r,s}$ on machine $M_{k'}$ and C_{\max} be the makespan before moving. We need to find the temporal position of $O_{i,j}$ on machine $M_{k'}$ so that C_{\max} is minimized. There are two possibilities

- 1) Insert $O_{i,j}$ before $O_{r,s}$: this can violate the critical path that contains $O_{r,s}$. If the stop time of $O_{i,j}$ on $M_{k'}$ is larger than the start time of $O_{i,j+1}$, it will also violate the critical path containing $O_{i,j+1}$.
- 2) Insert $O_{i,j}$ after critical block B : this only violates the critical path that contains $O_{i,j+1}$.

Let $\text{st}(O_{i,j})$ be the start time of $O_{i,j}$ on machine $M_{k'}$. It is equal to the maximum value of $\text{pest}(O_{i,j})$ and the completion time of $O_{p,q}$. Let $\text{stop}(B)$ be the stop time of critical block B . We introduce two values: *positionValue1* and *positionValue2* to evaluate the increasing amounts of C_{\max} for the two positions described before, where

$$\text{positionValue1} = \max(\text{st}(O_{r,s}) - (\text{st}(O_{i,j}) + p_{i,j,k'}),$$

$$\text{st}(O_{i,j+1}) - (\text{st}(O_{i,j}) + p_{i,j,k'}))$$

$$\text{positionValue2} = \text{stop}(B) + p_{i,j,k'} - \text{st}(O_{i,j+1}).$$

Theorem 11 (Reducing New Makespan Value): If $\text{positionValue1} \leq \text{positionValue2}$, inserting $O_{i,j}$ before $O_{r,s}$ can reduce the amount of increase in C_{\max} than by inserting $O_{i,j}$ after critical block B . Otherwise, inserting $O_{i,j}$ after critical block B can reduce the amount of increase in C_{\max} .

Proof: When inserting $O_{i,j}$ before $O_{r,s}$, the lower bound of the new makespan is $C_1 = C_{\max} + \text{positionValue1}$. When inserting $O_{i,j}$ after critical block B , the lower bound of new makespan is $C_2 = C_{\max} + \text{positionValue2}$.

If $\text{positionValue1} \leq \text{positionValue2}$, then $C_1 \leq C_2$: we choose to insert $O_{i,j}$ before $O_{r,s}$ instead of inserting $O_{i,j}$ after critical block B . Otherwise, inserting $O_{i,j}$ after critical block B is selected instead of inserting $O_{i,j}$ before $O_{r,s}$.

From the results of Theorems 7–11, it is seen that the best moves of GLS can be found in two steps.

First, we identify a set of promising operations S so that when moving each operation in S to another machine, one or more objectives described in Section II can be minimized. Next, when moving $O_{i,j}$ in set S to a new machine $M_{k'}$, we find a suitable position in an array of ordered operations on $M_{k'}$ to insert $O_{i,j}$ so that the makespan of the new schedule is minimized. Theorems 9 and 10 have shown that the improvement of *makespan* can only be obtained by moving operations on the *critical paths* of a schedule. Therefore, in order to reduce the computational cost, only operations that belong to the *critical paths* are selected to be moved.

The GLS procedure for finding promising operations of the current schedule is described as follows.

- Step 1) Find all operations on critical paths to insert to set U .
- Step 2) For each operation $O_{i,j}$ in U with processing time $p_{i,j,k}$ on machine M_k , identify all other machines that can process $O_{i,j}$. Insert $O_{i,j}$ and an alternative machine $M_{k'}$ to set V if one of the following conditions is satisfied.
 - a) The new processing time of $O_{i,j}$ on $M_{k'}$, $p_{i,j,k'}$ is smaller than $p_{i,j,k}$ (to improve total tardiness).
 - b) M_k is a machine whose workload is equal to the critical machine workload value MW and the new workload on machine $M_{k'}$ where $O_{i,j}$ is inserted is smaller than MW (to improve critical workload).
 - c) There is only one critical path. The new processing time is $p_{i,j,k}$, and $pest(O_{i,j})$ is smaller than the start time of $O_{i,j}$ on machine M_k (critical machine workload and total workload are unchanged but the *makespan* can be improved).

For each operation $O_{i,j}$ and its corresponding machine $M_{k'}$ in set V , the procedure to find its suitable position on machine $M_{k'}$ is described as follows.

- Step 3) Remove operation $O_{i,j}$ with its corresponding machine $M_{k'}$ from set V .
- Step 4) If there are no operations in machine $M_{k'}$, insert $O_{i,j}$ to $M_{k'}$. Otherwise, find a position of an operation $O_{p,q}$ in $M_{k'}$ so that $pest(O_{i,j})$ is smaller than the current start time of $O_{p,q}$.
- Step 5) If $pest(O_{i,j})$ plus the processing time of $O_{i,j}$ on machine $M_{k'}$ ($p_{i,j,k'}$) is at most the start time of the next operation of $O_{p,q}$ (that is $O_{r,s}$) on machine $M_{k'}$, then insert $O_{i,j}$ after $O_{p,q}$.
- Step 6) If $O_{r,s}$ is *not* on a critical path, insert $O_{i,j}$ after $O_{p,q}$. Otherwise, if $O_{i,j+1}$ is not on critical path, insert $O_{i,j}$ after critical block B . If both $O_{i,j+1}$ and $O_{r,s}$ are on critical paths, apply Theorem 11 to find the best move.
- Step 7) Stop if there are no operations left in set v , else go to Step 3.

At the end of Step 7, we will obtain a set of neighboring schedules of the current schedule. The encodings of these schedules are used to update the current population.

B. Descriptions of MOEA-GLS

We adopt the operation order machine selection (OOMS) chromosomes for FJSPs from [8]. This chromosome has two

parts: the operation order part is integer-based encoded, whereas the machine selection part is binary encoded for identifying selected machines. Please refer to [8] for the decoding method to evaluate its *makespan*, critical machine workload and the total workload. We also use the elitism memory introduced in [13] and [21] to keep non-dominated solutions. The MOEA-GLS algorithm (see Fig. 2) is described in detail as follows.

- Step 1) *Initialization*: generate an initial population using composite dispatching rules.
- Step 2) *Evaluation*: combine the current generation with solutions from the elitism memory. Applying fast non-domination and crowding distance from [20] to evaluate the results. Then, only n best solutions are used to update the current population.
- Step 3) *Selection*: apply ranking selection to generate the offspring population. Randomly select two operations A and B from the solutions of Step 2. If A dominates B , copy A to offspring population and vice versa. If A and B are non-dominated, their bigger crowding value is used to select the best one to insert to the offspring population.
- Step 4) *Crossover*: apply two-point crossover for OOMS chromosomes [8] with crossover probability p_c in offspring population.
- Step 5) *Mutation*: apply mutation for OOMS chromosomes [8] with crossover probability p_m in offspring population.
- Step 6) *Guided local search*:
 - a) Decode offspring population, apply fast nondomination in [20] to rank the offspring population, select $x\%$ of best solutions of offspring population for performing GLS as described in Section IV-A.
 - b) Decode the solutions of GLS, apply fast nondomination in [20] to rank them, replace maximum $y\%$ number of the worst solutions in offspring population by the best solutions of GLS.
- Step 7) *Update the elitism memory*: select the solutions with rank 1 in offspring population to update the elitism memory. If a solution C dominates any results in elitism memory, delete these results and copy C to elitism memory. If C is non-dominated by all results in the elitism memory, C is also copied to elitism memory.
- Step 8) Return to Step 2.

This algorithm terminates when it reaches a predefined number of populations. In order to reduce the computational time for MOEA-GLS, in the local search at Step 6(a), only $x\%$ of the best solutions of offspring population are selected for GLS. To keep the diversity of the next population, we do not replace all solutions of offspring population by the results of GLS. Only a maximum of $y\%$ of the number of the worst solutions in the offspring population are replaced by the best solutions in Step 6(b). x and y are two predefined parameters. Note that x can be determined using predefined numbers of chromosomes in one population, while y is an upper limit. In this paper, the values of x and y were estimated based on preliminary computational experiments as $x = 33.3\%$ and $y = 50\%$ (see Section V).

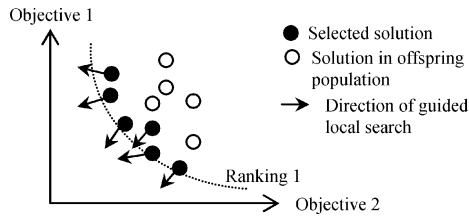


Fig. 5. Selection of the best solutions in the offspring population for GLS.

An illustration of the selection of $x\%$ of the best solutions in offspring population for GLS is given in Fig. 5.

In this algorithm, all non-dominated solutions are stored in a separate elitism memory. For some MOEA algorithms (e.g., SPEA [21]), the size of the elitism memory has been restricted due to the memory storage and computational time. In our algorithm, we did not find many non-dominated solutions for FJSP (see Section V). Therefore, the restriction on the size of elitism memory was not necessary.

VI. EXPERIMENTAL RESULTS

In order to evaluate the efficacy and performance of the MOEA-GLS algorithm proposed in this paper, six popular benchmarks [6], [7], [11] are used (represented by n jobs \times m machines). In order to evaluate single lower bound values and multi-lower bound set for each benchmark, the algorithms described in Section IV are applied.

Our problem sets were distinguished as follows: Test sample I for the small- and medium-sized instances without release dates (all release dates of jobs are 0), Test sample II for the small- and medium-sized instances with release dates, Test sample III for the large-sized instances without release dates. The analysis of the obtained results will be presented in Section VI-C. The system was implemented using C++, running on a 2-GHz PC with 512-MB RAM. The best results and average processing time of MOEA-GLS after 30 runs were reported.

Through experimentation, suitable parameter values were chosen; namely, population size 200, crossover probability 0.8, mutation probability 0.3, number of generations 200, percentage of best results were used for GLS: 33.3%, maximum percentage of best results of GLS that are used to update current generation: 50%.

For each instance, the obtained results are reported in two tables. The first table presents single lower bound values of three objectives: F_1^* (makespan), F_2^* (critical machine workload), F_3^* (total workload), and multiset lower bounds from the B&B algorithm described in Section IV. The second table compares the results obtained by our algorithm MOEA-GLS with the results of approach by localization and controlled genetic algorithms (AL-CGA) by Kacem *et al.* [6], [7], the results of particle swarm optimization and simulated annealing (PSO-SA) from Xia and Wu [11], and the results of TS from [29] for solving the same instance. In the same table, the results shown in boldfaced font *dominates* the results in italic font. The results of MOEA-GLS for each instance are then used to initialize the B&B algorithm described in Section IV-B. The multi-lower bound set is then reported.

TABLE I
LOWER BOUNDS FOR 8 JOBS \times 8 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	11	10	73	0.02 seconds
Multi Set	14	13	73	57.45 seconds
	11	11	77	

TABLE II
COMPARISON OF RESULTS ON 8 JOBS \times 8 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
AL-CGA	15	x^a	79
	16	x^a	75
PSO-SA	16	13	73
	15	12	75
MOEA-GLS	16	13	73
	15	12	75
	14	12	77
	16	11	77

^aThe symbol x indicates that the authors did not provide the result of this objective.

TABLE III
LOWER BOUNDS FOR 10 JOBS \times 10 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	7	5	41	0.02 seconds
Multi Set	8	7	41	19.77 seconds
	8	5	42	
	7	5	43	
	7	6	42	

TABLE IV
COMPARISON OF RESULTS ON 10 JOBS \times 10 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
AL-CGA	8	7	41
	8	5	42
	7	5	<i>45^b</i>
PSO-SA	7	6	<i>44^b</i>
	8	7	41
MOEA-GLS	8	5	42
	7	5	43^b
	7	6	42^b

^bThe results in boldfaced font dominate the results in italic font.

A. Test Sample I

This test sample compares the performances of MOEA-GLS, AL-CGA [6], [7], and PSO-SA [11] on three small- and medium-sized FJSPs without release dates (or that the release dates are set to 0 for all jobs). These FJSPs were presented by Kacem *et al.* [6], [7].

1) *Problem 8 \times 8*: Number of nodes explored by B&B: 1.637.084 (see Table I).

Average processing time of MOEA-GLS: 9.097 s (see Table II).

2) *Problem 10 \times 10*: Number of nodes explored by B&B: 449.460 (see Table III).

Average processing time of MOEA-GLS: 16.647 s (see Table IV).

TABLE V
LOWER BOUNDS FOR 15 JOBS \times 10 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	10	10	91	0.03 seconds
Multi Set	11	10	93	6.38
	11	11	91	hours

TABLE VI
COMPARISON OF RESULTS ON 15 JOBS \times 10 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
PSO-SA	12	11	91
MOEA-GLS	11	10	93
	11	11	91

TABLE VII
LOWER BOUNDS FOR 4 JOBS \times 5 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	16	7	32	0.01 seconds
Multi Set	16	8	32	0.02
	16	7	33	seconds

TABLE VIII
COMPARISON OF RESULTS ON 4 JOBS \times 5 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
AL-CGA	18	8	32
	18	7	33
	16	9	35
	16	10	34
MOEA-GLS	16	8	32
	16	7	33

3) *Problem 15 \times 10*: Number of nodes explored by B&B: 695.342.910 (see Table V).

Average processing time of MOEA-GLS: 24.149 s (see Table VI).

B. Test Sample II

This test sample compares the performances of MOEA-GLS, AL-CGA [6], [7], and PSO-SA [11] on three small- and medium-sized FJSPs with release dates for each job. These FJSPs were presented by Kacem *et al.* [6], [7].

1) *Problem 4 \times 5*: Release dates: $r_1 = 3, r_2 = 5, r_3 = 1, r_4 = 6$.

Number of nodes explored by B&B: 115 (see Table VII).

Average processing time of MOEA-GLS: 9.314 s (see Table VIII).

2) *Problem 10 \times 7*: Release dates: $r_1 = 2, r_2 = 4, r_3 = 9, r_4 = 6, r_5 = 7, r_6 = 5, r_7 = 7, r_8 = 4, r_9 = 1, r_{10} = 0$.

Number of nodes explored by B&B: 83.538 (see Table IX).

Average processing time of MOEA-GLS: 13.564 s (see Table X).

3) *Problem 15 \times 10*: Release dates: $r_1 = 5, r_2 = 3, r_3 = 6, r_4 = 4, r_5 = 9, r_6 = 7, r_7 = 1, r_8 = 2, r_9 = 9, r_{10} = 0, r_{11} = 14, r_{12} = 13, r_{13} = 11, r_{14} = 12, r_{15} = 5$.

TABLE IX
LOWER BOUNDS OF 10 JOBS \times 7 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	15	9	60	0.02 seconds
Multi Set	15	11	61	3.13 seconds
	16	12	60	
	15	10	62	

TABLE X
COMPARISON OF RESULTS ON 10 JOBS \times 7 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
AL-CGA	15	11	61
	16	12	60
	16	10	66
	17	10	64
MOEA-GLS	18	10	63
	15	11	61
	16	12	60
	15	10	62

TABLE XI
LOWER BOUNDS OF 15 JOBS \times 10 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	23	10	91	0.02 seconds
Multi Set	23	11	91	3.36
	23	10	93	hours

TABLE XII
COMPARISON OF RESULTS ON 15 JOBS \times 10 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
AL-CGA	24	11	91
	23	11	95
MOEA-GLS	23	11	91
	23	10	93

Number of nodes explored by B&B: 374.344.900 (see Table XI).

Average processing time of MOEA-GLS: 17.509 s (see Table XII).

C. Test Sample III

In this test sample, we randomly select three large-sized FJSPs from [29] where the number of operations is up to 300. Note that in these instances, the probabilities of machines that can process one operation are from $(1/2)m$ to $(4/5)m$ where m is number of machines [29]. The best makespan results of TS from [29] are also reported.

1) *Problem 20 \times 10 (la30)*: Average processing time of MOEA-GLS: 35.18 min (see Tables XIII and XIV).

2) *Problem 30 \times 10 (la35)*: Average processing time of MOEA-GLS: 5.92 min (see Tables XV and XVI).

3) *Problem 15 \times 15 (la40)*: Average processing time of MOEA-GLS: 15.48 min (see Tables XVII and XVIII).

TABLE XIII
LOWER BOUNDS FOR 20 JOBS \times 20 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	1068	1068	10680	0.14 seconds

TABLE XIV
UPPER BOUNDS OF RESULTS ON 20 JOBS \times 10 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
TS	1070	x	x
	1075	1075	10680
MOEA-GLS	1077	1073	10680
	1079	1072	10680

TABLE XV
LOWER BOUNDS OF 30 JOBS \times 10 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	1549	1549	15485	0.01 seconds

TABLE XVI
UPPER BOUNDS OF RESULTS ON 30 JOBS \times 10 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
TS	1551	x	x
MOEA-GLS	1550	1550	15485

TABLE XVII
LOWER BOUNDS OF 15 JOBS \times 15 MACHINES

Lower Bound	Makespan	Critical Machine Workload	Total workload	Time
Single Values	955	765	11472	0.03 seconds

D. Analysis of Experimental Results

The results of the three test samples indicate that the MOEA-GLS algorithm can obtain good solutions in short computational time. All of the results of MOEA-GLS are very close to the single lower bound provided for each objective. In many solutions, two out of three objectives reach the single lower bounds. The number of nodes explored by B&B is also small due to high-quality upper bound solutions and single lower bound values.

In instances of test sample I, for 8 jobs \times 8 machines, although the MOEA-GLS fails to obtain solutions that dominate the solutions of AL-CGA and PSO-SA, it achieves a wider range of non-dominated solutions. For 10 jobs \times 10 machines, two new results of MOEA-GLS dominate the results from AL-CGA and PSO-SA. The multi-lower bound set described in Table III indicates that the results from MOEA-GLS are *Pareto*-optimal solutions. Xia and Wu [11] applied the PSO-SA to solve the 15 jobs \times 10 machines without release date. The solution obtained by MOEA-GLS (11 11 91) dominates the solution from PSO-SA (12 11 91). In these two new results of MOEA-GLS, two objectives have already reached the lower bounds.

TABLE XVIII
UPPER BOUNDS OF RESULTS ON 15 JOBS \times 15 MACHINES

Algorithms	Makespan	Critical Machine Workload	Total workload
TS	955	x	x
	955	783	11472
	957	780	11472
MOEA-GLS	963	779	11472
	964	777	11472
	966	775	11472

In instances of test sample II, for 4 jobs \times 5 machines, two results from MOEA-GLS dominate all the results from AL-CGA. For 10 jobs \times 7 machines, one new result from MOEA-GLS (15 10 62) dominates three solutions from AL-CGA. Similarly, for 15 jobs \times 10 machines, two results from MOEA-GLS dominate all the results from AL-CGA. More importantly, the multi-lower bound sets in Tables VII, IX, and XI indicate that the solutions obtained by MOEA-GLS are *Pareto* optimal.

In test sample III, for each instance, we reported the best makespan results of [29]. Note that in [29], Hurink *et al.* applied TS to minimize only a single objective (makespan), the remaining objectives were not reported. MOEA-GLS outperforms TS in solving the instance of 30 jobs \times 10 machines. It obtains the same *optimal* makespan with TS in the instance of 15 jobs \times 15 machines, while TS obtains a better result than MOEA-GLS in the instance of 20 jobs \times 10 machines.

Besides the good solutions discovered by MOEA-GLS, its processing time is also acceptable. The maximum computational time for 15 jobs \times 10 machines in test sample I is around 25 s. Although MOEA-GLS is worst than TS in solving hard problems such as 20 jobs \times 10 machines, it gets the result near to the result from TS in shorter computational time (35.18 min), while TS requires 38.18 h to finish computing the same instance on a sun 4/20 workstation. Ishibuchi *et al.* [13] mentions that expensive computational time is a significant issue in the integration of local search to EAs. However, as presented in test samples I, II, and III, this problem can be solved in our algorithm using guidance for local search. GLS explores the areas where new solutions dominate or are of the same rank (non-dominated) with selected solutions. Therefore, better solutions can be achieved by GLS. Furthermore, the restriction of selecting only the best results for applying GLS also helps reduce the computational cost. In each generation, only a predefined number of the best solutions are selected for applying GLS. It provides more diversity toward *Pareto*-optimal solutions.

VII. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed an efficient algorithm for solving multi-objective FJSPs. Instead of applying random local search methods to improve the results in each generation (e.g., [11] and [13]), we introduced a GLS procedure to accelerate the process of convergence to *Pareto*-optimal solutions. It not only searches unexplored space to find better solutions, but also reduces the computational time due to the selection of the best moves. Lower bounds for evaluating the efficacy of the algorithms for solving FJSPs are also introduced. Empirical

studies show that the gaps between the obtained results and single lower bounds are very small. Furthermore, the results obtained by MOEA-GLS frequently dominate the results of previous researchers in solving the same benchmarks. From the multi-lower bound set generated by the B&B algorithm, it indicates that many results obtained by MOEA-GLS are *Pareto*-optimal solutions. It is shown to be capable of obtaining high quality, wide range of *Pareto*-optimal solutions, and efficient performance on overall benchmark problems.

More comprehensive studies can be applied to extend the MOEA-GLS. Different FJSP data with bigger sizes will be investigated. Other possible criteria in multi-objective optimization will be considered. Furthermore, more local search methods will be analyzed to integrate to the MOEA-GLS algorithm.

APPENDIX

Detailed processing times of all operations of the instances as well as their Gantt charts that were presented in Section VI are available at: http://www.ntu.edu.sg/home/asjctay/doc/MOEA_GLS.pdf.

REFERENCES

- [1] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *Eur. J. Oper. Res.*, vol. 113, pp. 390–434, 1999.
- [2] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flow shop and job-shop scheduling," *Math. Oper. Res.*, vol. 1, pp. 117–129, 1976.
- [3] M. Pinedo, *Scheduling Theory, Algorithms, and Systems*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2002, ch. 2.
- [4] Y. Mati and X. Xie, "The complexity of two-job shop problems with multi-purpose unrelated machines," *Eur. J. Oper. Res.*, vol. 152, pp. 159–169, 2004.
- [5] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Manag. Sci.*, vol. 35, pp. 164–176, 1989.
- [6] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Trans. Syst. Man Cybern. C, Appl. Rev.*, vol. 32, no. 1, pp. 1–13, Feb. 2002.
- [7] I. Kacem, S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic," *Math. Comput. Simul.*, vol. 60, pp. 245–276, 2002.
- [8] N. B. Ho, J. C. Tay, and E. M. K. Lai, "An effective architecture for learning and evolving flexible job-shop schedules," *Eur. J. Oper. Res.*, vol. 179, no. 2, pp. 316–333, 2007.
- [9] N. B. Ho and J. C. Tay, "Evolving dispatching rules for solving the flexible job-shop problem," in *Proc. IEEE Congr. Evol. Comput. (CEC2005)*, Sep., vol. 3, pp. 2848–2855.
- [10] J. C. Tay and D. Wibowo, "An effective chromosome representation for evolving flexible job shop schedules," in *Proc. Genetic Evol. Comput.*, 2004, vol. 3103, pp. 210–221.
- [11] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Comput. Ind. Eng.*, vol. 48, pp. 409–425, 2005.
- [12] M. Mastrolilli and L. M. Gambardella, "Effective neighborhood functions for the flexible job shop problem," *J. Sched.*, vol. 3, pp. 3–20, 2000.
- [13] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multi-objective permutation flowshop scheduling," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 204–223, Apr. 2003.
- [14] E. K. Burke and S. J. D. Landa, "The Design of Memetic Algorithms for Scheduling and Timetabling Problems," in *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, vol. 166, W. Hart, N. Krasnogor, and J. Smith, Eds. New York: Springer-Verlag, 2004, pp. 289–312.
- [15] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. New York: Wiley, 2001.
- [16] P. Brucker, B. Jurisch, and A. Krämer, "Complexity of scheduling problems with multi-purpose machines," *Ann. Oper. Res.*, vol. 70, pp. 57–73, 1997.
- [17] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Ann. Discr. Math.*, vol. 5, pp. 287–236, 1979.
- [18] C. A. C. Coello, "Recent Trends in Evolutionary Multi-objective Optimization," in *Evolutionary Multi-objective Optimization: Theoretical Advances and Applications*, L. Jain, R. Goldberg, and A. Abraham, Eds. London, U.K.: Springer-Verlag, 2005, pp. 7–32.
- [19] J. D. Schaffer, "Multiple-objective optimization with vector evaluated genetic algorithms," in *Proc. 1st Int. Conf. Genetic Algorithms*, 1985, pp. 93–100.
- [20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 181–197, Apr. 2002.
- [21] E. Zitzler and L. Thiele, "Multi-objective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.
- [22] J. D. Knowles and D. W. Corne, "Memetic Algorithms for Multi-objective Optimization: Issues, Methods and Prospects," in *Recent Advances in Memetic Algorithms*, W. E. Hart, N. Krasnogor, and J. E. Smith, Eds. New York: Springer-Verlag, 2005, pp. 313–352.
- [23] N. Krasnogor and J. E. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.
- [24] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Trans. Syst. Man, Cybern. C, Appl. Rev.*, vol. 28, no. 3, pp. 392–403, Aug. 1998.
- [25] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discr. Appl. Math.*, vol. 49, pp. 107–127, 1994.
- [26] J. Carlier and E. Pinson, "Adjustment of heads and tails for the job-shop problem," *Eur. J. Oper. Res.*, vol. 78, pp. 146–161, 1994.
- [27] M. Haouari and A. Gharbi, "Lower bounds for scheduling on identical parallel machines with heads and tails," *Ann. Oper. Res.*, vol. 129, pp. 187–204, 2004.
- [28] J. D. Knowles and D. W. Corne, "Approximating the non-dominated front using the pareto archived evolution strategy," *Evol. Comput.*, vol. 8, pp. 149–172, 2000.
- [29] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multipurpose machines," *Or Spektrum*, vol. 15, pp. 205–215, 1994.



Nhu Binh Ho received the B.Eng. degree in computer engineering from the Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam, in 2001, and the Ph.D. degree in computer science from Nanyang Technological University, Singapore, Singapore, in 2007.

He is currently with the Evolutionary and Complex Systems Program, School of Computer Engineering, Nanyang Technological University. His current research interests include evolutionary algorithms, operations research, and scheduling algo-

gorithms in manufacturing.



Joe Cing Tay (S'94-M'02-SM'06) received the Ph.D. degree from Nanyang Technological University (NTU), Singapore, Singapore, in the field of constraint optimization in 2000.

He is currently the Chief Scientist for ROSS Scientific Pte. Ltd., Singapore, and an Assistant Professor at the School of Computer Engineering (SCE), Nanyang Technological University, where he is also the Director for the Evolutionary and Complex Systems Programme. He was earlier a Senior Consultant for ILOG Singapore Pte. Ltd., where he was special-

izing in optimization and rule-based systems. His current research interests include computational immunology and epidemiology, discrete optimization, evolutionary computation, complex dynamical system simulation, and multiagent learning models. He is the author or coauthor of several papers published in leading conferences and journals. He is an Editorial Board Member of the *Journal Future Generation Computer Systems (FGCS)*, Elsevier Science.

Dr. Tay was awarded an Outstanding Continental Field Applications Engineer Award, the Tan Chin Tuan Fellowship in Engineering at the Centre for Mathematics and Physics for life sciences and experimental biology at the University College London. He is a member of the Association for Computing Machinery.