Using Evolutionary Computation and Local Search for Solving Multi-objective Flexible Job Shop Problems

Nhu Binh Ho
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 639798
honhubinh@pmail.ntu.edu.sg

Joc Cing Tay
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 639798
asjctay@ntu.edu.sg

ABSTRACT

Finding realistic schedules for Flexible Job Shop Problems has attracted many researchers recently due to its NP-hardness. In this paper, we present an efficient approach for solving the multi-objective flexible job shop by combining Evolutionary Algorithm and Guided Local Search. Instead of applying random local search to find neighborhood solutions, we introduce a guided local search procedure to accelerate the process of convergence to *Pareto*-optimal solutions. The main improvement of this combination is to help diversify the population towards the Pareto-front. Empirical studies show that 1) the gaps between the obtained results and known lower bounds are small, and 2) the multi-objective solutions of our algorithms dominate previous designs for solving the same benchmarks while incurring less computational time.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods and Search**]: Scheduling

General Terms

Algorithms, Scheduling, Optimization, Search.

Keywords

Multi-objective Evolutionary Algorithm, Guided Local Search, Flexible Job Shop Problems.

1. 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). 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 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 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 Job-Shop Scheduling Problem (FJSP). It extends the definition of the JSP by allowing an operation to be processed without interruption on one of a set of pre-defined 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 2 machines for minimizing *makespan* (the maximum completion time of all operations), 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 [5] can guarantee optimal solutions but are computationally expensive even for small sized problems. Other approximation and randomized algorithms methods such as Evolutionary Algorithms [6-8], Simulated Annealing [9], or Tabu Search [10] have demonstrated an efficacy for solving FJSPs.

The combination of Evolutionary Algorithms (EAs) and Local Search methods (known as *Memetic Algorithms*) for solving scheduling problems has been increasingly studied by researchers and good solutions [11, 12] have been obtained. The local search methods by themselves have also achieved promising results [9, 10]. 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 Evolutionary Algorithm with Guided Local Search (MOEA-GLS) algorithm to solve multi-objective FJSPs. The algorithm comprises three parts: MOEA, Guided Local Search and Elitism Memory. Instead of simply using a randomized local search for finding neighborhoods of a schedule, the local search part in every generation is intelligently guided by an efficient selection towards promising areas. These mechanisms guarantee that only the moves that obtain better solutions are visited. Therefore, more good quality solutions can be obtained and computational cost 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, 9]. 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 2 gives the formal definition of the FJSP. Section 3 reviews recent related works for solving the FJSP and classifies current approaches for solving multi-objective optimization problems using EAs. Section 4 describes Guided Local Search and the integration of MOEA and Guided Local Search. It also provides some theoretical considerations and the procedure to find the best moves. Section 5 analyzes the performance results of MOEA-GLS when applied to solve common benchmarks in literature. Finally, Section 6 gives some concluding remarks and directions for future work.

2. PROBLEM DEFINITION

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

- Let $J = J_{1\le i\le n}$, indexed i, be a set of n jobs to be scheduled.
- Each job J_i consists of a predetermined sequence of operations. Let $O_{i,j}$ be operation j of J_i.
- Let $M = \{M_k\}_{1\le k\le m}$, indexed k, be a set of m machines.
- Each machine can process only one operation at a time.
- 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. Three objectives (which have been used to evaluate the efficacy of other algorithms in solving multi-objective FJSPs [6, 7, 9]) are used in this paper; namely,

- Minimization of overall completion time (*makespan)*: $F_1 = \max\{C_i \mid i = 1,..,n\}$.
- Minimization of critical machine workload: $F_2 = \max\{W_k \mid k = 1,..,m\}$
- 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 [13].

The FJSP can also be considered to be a Multi-Purpose Machine (MPM) job-shop [14]. Using the α|β|γ notation of Graham *et al.* [15], the problem we wish to solve can be denoted by:

$$J\ MPM\ \mid prec\ r_j \mid\ F_1\ F_2\ F_3$$

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 finally F_3 represents the total workload of all machines.

In this paper, we will assume that:

- All machines are available at time 0.
- Each job has its own release date.
- The order of operations for each job is predefined and invariant.
- The machine can execute only one operation at a time.

3. LITERATURE REVIEW

Evolutionary Algorithms (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 [16].

Population-based approaches, such as VEGA [17], 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, 11]. 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 [16].

Pareto-based approaches, also known as the second generation of MOEAs [16], use the concept of domination to find the 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 nondominated 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 literature. The most popular methods are NSGA-II [18] and SPEA [19] 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 towards the Pareto front of the problems.

Recently, many researchers have studied the combination of EA and Local Search for solving multi-objective scheduling problems. The general idea is to enhance solutions obtained by each generation by local search to get better results. Subsequently, these obtained results are used to update the quality of next generation. Ishibuchi *et al.*[11] combine EA and Local Search for solving 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 pre-defined 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 [9] use 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 above also face issues of finding suitable weight vectors to diversify the results towards 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 that an improvement in the obtained results. The difference between this work and previous approaches on solving multi-objective FJSPs is that we use a mechanism of guided local search 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].

4. THE MOEA-GLS ALGORITHM

In this Section, the detailed description of the MOEA-GLS is presented. Section IV.A gives common definitions that are used in this paper. Theoretical considerations for improving Guided Local Search are also discussed. Section IV.B presents our approach to integrate Guided Local Search with MOEA and the selection of designed parameters to reduce computational time.

4.1 Guided Local Search

For a given schedule of the FJSP, a neighborhood can be obtained by moving an operation from one machine to another machine. However, not all moves improve the current results and a complete enumeration of the neighbourhood 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 introduced as follows to be used throughout this paper.

Definition 1 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 2 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.

Definition 3 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 4 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 1 (*minimizing the total workload*):

When moving an operation $O_{i,j}$ from machine M_k to another machine $M_{k'}$, the new total workload (*TW*) 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:

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

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:

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

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

Theorem 2 (*minimizing the critical machine workload*):

When moving an operation $O_{i,j}$ from one of most loaded machines (e.g. M_k) to the other machine $M_{k'}$ on a schedule, the critical machine workload is minimal (e.g. at most the critical machine workload *MW*) if the workload on machine $M_{k'}$ *plus* 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 3:

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

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

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

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 $WL'(M_{k'}) \le WL(M_k)$ or $WL(M_{k'}) + p_{i,j,k'} \le MW$, 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 3 (*minimizing makespan*):

The *makespan* of a schedule cannot be reduced when moving operations that are not on the *critical paths* to other machines.

Proof: The *makespan* is the total processing time of all operations of a 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 4 (*minimizing 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 in case of new processing time $p_{i,j,k'}$ 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 operation $O_{i,j-1}$ ($pest(O_{i,j})$). Using the constraint of $pest(O_{i,j})$, we identify the operation $O_{p.q}$ in machine $M_{k'}$ which has a stop time that is smaller or equal to $pest(O_{i,j})$. $O_{i,j}$ can 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 Figure 1 below.

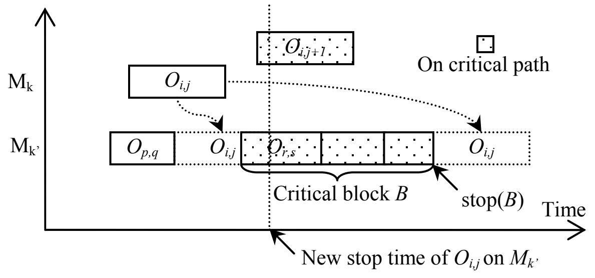

Figure 1. Finding suitable position to insert $O_{i,j}$ to machine $M_{k'}$.

In Figure 1, both the immediate successor operations $O_{i,j+1}$ 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 position of $O_{i,j}$ on machine $M_{k'}$ so that i 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 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 $st(O_{i,j})$ be start time of $O_{i,j}$ on machine $M_{k'}$. It is equal to maximum value of $pest(O_{i,j})$ and completion time of $O_{p,q}$. Let $stop(B)$ be stop time of critical block B. We introduce two values: *positionValue1* and *positionValue2* to evaluate the increasing amount of C_{max} for two positions described above:

$$positionValue1 = \max(\ st(O_{r,s}) - (st(O_{i,j})+p_{i,j,k'}),$$
$$st(O_{i,j+1}) - (st(O_{i,j})+p_{i,j,k'}))$$

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

Theorem 5 (*reducing new makespan value*):

If *positionValue1* ≤ *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 new *makespan* is: $C_1 = C_{max} + positionValue1$. When inserting $O_{i,j}$ after critical block B, the lower bound of new *makespan* is: $C_2 = C_{max} + positionValue2$.

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

From the results of Theorem 1 to Theorem 5, it is seen that the best moves of Guided Local Search can be found into two steps. *Firstly*, 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 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 new schedule is minimized. Theorem 3 and Theorem 4 have shown that the improvement of *makespan* can only be obtained by moving operations on *critical paths* of a schedule. Therefore, in order to reduce the computational cost, only operations that belong to *critical paths* are selected to be moved.

The Guided Local Search 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 $p_{i,j,k}$ in machine M_k, identify all machines that can process $O_{i,j}$ except M_k. Inserting $O_{i,j}$ and an alternative machine $M_{k'}$ to set V if one of the following conditions is satisfied:

a) The new processing time $p_{i,j,k'}$ on machine $M_{k'}$ is smaller than $p_{i,j,k}$ (to improve total tardiness).

b) M_k is a machine whose workload is equal to 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 (or only one machine that its stop time is latest), the new processing time is equal to $p_{i,j,k}$, and $pest(O_{i,j})$ is smaller than start time of $O_{i,j}$ on machine M_k (critical machine workload and total workload are unchanged but *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 is no operation 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 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 $M_{k'}$, 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 a 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 5 to find the best move.

Step 7) Stop if there are no operations remaining in set V. Otherwise, go back to Step 3).

At the end of Step 7), we will obtain a set of neighborhood schedules of the current schedule. The encodings of these schedules are then used to update the current population. Note that after Step 2), only the schedules that their neighborhood schedules dominate or are at least of the same rank with them are selected for exploitation.

4.2 Description of MOEA-GLS

The overall control flow of the MOEA-GLS algorithm is summarized in Figure 2. To generate a good and diversified initial population, we employ Composite Dispatching Rules presented in [8]. 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 Guided Local Search module uses the procedure described in Section IV.A. Coello [16] mentions that the second generation of MOEAs have introduced the use of elitism selection strategy. Similar to previous research in MOEA [11, 19, 20], 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.

We adopt the Operation Order Machine Selection (OOMS) chromosomes for FJSPs from [8]. This chromosome has two parts: the Operation Order part is integer 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 total workload. We also use the elitism memory introduced in [11, 19] to keep non-dominated solutions. The MOEA-GLS algorithm is described as follows:

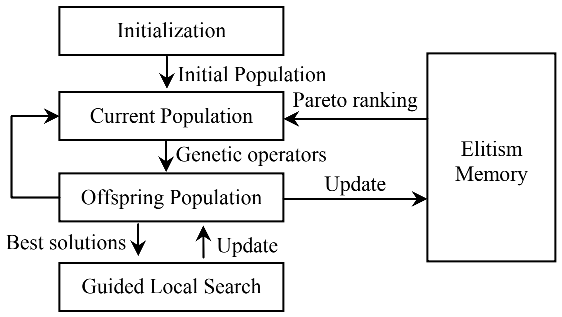

Figure 2. The outline of the MOEA-GLS algorithm

Step 1) *Initialization*: generate an initial population using Composite Dispatching Rules.

Step 2) *Evaluation*: combine current generation with the solutions in elitism memory. Applying fast non-domination and crowding distance from [18] to evaluate the results. Then, only *n* best solutions are used to update current population.

Step 3) *Selection*: apply ranking selection to generate offspring population. Randomly select two operations *A* and *B* in 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 offspring population.

Step 4) *Crossover*: apply 2-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 non-domination in [18] to rank the offspring population, select *x%* of best solutions of offspring population for performing guided local search described in Section IV.A.

b) Decode the solutions of guided local search, apply fast non-domination in [18] to rank them, replace maximum *y%* number of the worse solutions in offspring population by the best solutions of guided local search.

Step 7) *Update elitism memory*: select the solutions with ranking 1 in offspring population to update elitism memory. If the solution *C* dominates any results in elitism memory, delete these results and copy *C* to elitism memory. If *C* is non-dominated with all results in elitism memory, *C* is also copied to elitism memory.

Step 8) Return to Step 2).

This algorithm terminates when it reaches a pre-defined 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 guided local search. To keep the diversity of the next population, we do not replace all solutions of offspring population by the results of guided local search. 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 pre-defined parameters. Note that *x* can be determined using pre-defined 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). An illustration of the selection of *x*% of the best solutions in offspring population for guided local search is given in Figure 3.

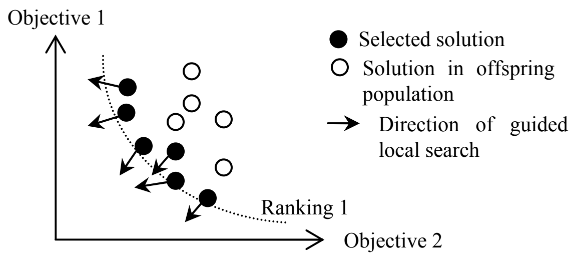
Objective 1
Selected solution
Solution in offspring population
Direction of guided local search
Ranking 1
Objective 2

Figure 3. Illustration of the selection of best solutions in offspring population to apply Guided Local Search.

In this algorithm, all non-dominated solutions are stored in a separate elitism memory. For some MOEA algorithms (e.g. SPEA [19]), the size of the elitism memory has been resized 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.

5. EXPERIMENTAL RESULTS

In order to evaluate the efficacy and performance of the MOEA-GLS algorithm proposed in this paper, six benchmark problems [6, 7, 9] are used (represented by *n jobs* x *m machines*). The lower bounds of each instance can be found in [7].

Our problem sets were distinguished as follows: Test sample I for the instances without release date (all release dates of jobs are 0), Test sample II for the instances with release date. The analysis of the obtained results will be presented in Section V.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. Details processing times of all operations of these instances as well as their Gantt charts presented in this section are available at:

http://www.ntu.edu.sg/home/asjctay/doc/MOEA_GLS.pdf.

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 Guided Local Search: 33.3%, maximum percentage of best results of Guided Local Search that are used to update current generation: 50%.

For each instance, lower bound of three objectives: F_1^* (*makespan*), F_2^* (critical machine workload), and F_3^* (total workload) are presented. In Test sample II, the release date of each job will be provided. Average processing time of each instance is also reported. Each table shows the results obtained by our algorithm MOEA-GLS, the results of Approach by Localization and Controlled Genetic Algorithms (AL-CGA) from Kacem *et al.* [6, 7], the results of Particle Swarm Optimization and Simulated Annealing (PSO-SA) from Xia and Wu [9] for solving the same instance. In the same table, the results in boldfaced font *dominate* the results in italic font.

5.1 Test sample I

5.1.1 *Problem 8x8*

Lower bound: $F_1^* = 13$, $F_2^* = 10$, $F_3^* = 73$.

Average processing time: 9.097 seconds.

Table 1. Comparison of Results on 8 jobs x 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

[a]The symbol *x* indicates that the authors didn't provide the result of this objective

5.1.2 *Problem 10x10*

Lower bound: $F_1^* = 7$, $F_2^* = 5$, $F_3^* = 41$

Average processing time: 16.647 seconds.

Table 2. Comparison of Results on 10 jobs x 10 machines

Algorithms	MAKESPAN	Critical Machine Workload	Total workload
AL-CGA	8	7	41
	8	5	42
	7	*5*	*45*[b]
PSO-SA	*7*	*6*	*44*[b]
MOEA-GLS	8	7	41
	8	5	42
	7	**5**	**43**[b]
	7	**6**	**42**[b]

[b]The results in boldfaced font dominate the results in italic font

5.1.3 *Problem 15x10*

Lower bound: $F_1^* = 11$, $F_2^* = 10$, $F_3^* = 91$

Average processing time: 24.149 seconds.

Table 3. Comparison of Results on 15 jobs x 10 machines

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

5.2 Test sample II

5.2.1 *Problem 4x5*

Release date: $r_1 = 3, r_2 = 5, r_3 = 1, r_4 = 6$

Lower bound: $F_1^* = 16$, $F_2^* = 7$, $F_3^* = 32$

Average processing time: 9.314 seconds.

Table 4. Comparison of Results on 4 jobs x 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**

5.2.2 *Problem 10x7*

Release date: $r_1 = 2, r_2 = 4, r_3 = 9, r_4 = 6, r_5 = 7, r_6 = 5, \quad r_7 = 1, r_8 = 4, r_9 = 1, r_{10} = 0$

Lower bound: $F_1^* = 15$, $F_2^* = 9$, $F_3^* = 60$

Average processing time: 13.564 seconds.

Table 5. Comparison of Results on 10 jobs x 7 machines

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

5.2.3 *Problem 15x10*

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

Lower bound: $F_1^* = 23$, $F_2^* = 10$, $F_3^* = 91$

Average processing time: 17.509 seconds.

Table 6. Comparison of Results on 15 jobs x 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**

5.3 Analysis of Results

The results of Test sample I and Test sample II indicate that the MOEA-GLS algorithm can obtain good solutions with low computational cost. All of the results of MOEA-GLS are very close to the lower bounds provided for each objective. In many solutions, two out of three objectives reach the single lower bounds.

In instances of Test sample I, for the 8 jobs x 8 machines instance, although the MOEA-GLS fails to obtain any solutions that dominate the solutions of AL-CGA and PSO-SA, it achieves a wider range of non-dominated solutions. For the 10 jobs x 10 machine instance, two new solutions of MOEA-GLS dominate the solutions from AL-CGA and PSO-SA. Xia and Wu [9] applied the PSO-SA to solve the 15 jobs x 10 machines without release dates. The solution obtained by MOEA-GLS (**11 11 91**) dominates the solution from PSO-SA (*12 11 91*). In these two new solutions of MOEA-GLS, two objectives have already reached the lower bounds.

In instances of Test sample II, for the 4 jobs x 5 machines instance, two results from MOEA-GLS dominate all the results from AL-CGA. Similarly, for 15 jobs x 10 machine instance, two results from MOEA-GLS again dominate all the results from AL-CGA. Furthermore, one solution from MOEA-GLS dominates three solutions obtained by AL-CGA.

Besides the good solutions provided by MOEA-GLS, its processing time is also acceptable. The maximum computational time for 15 jobs x 10 machines in Test sample I is around 25 seconds. Ishibuchi *et al.* [11] mentions that expensive computational time is a significant issue in the integration of local search to evolutionary algorithms. However, as presented in Test sample I and Test sample II, this problem can be solved in our algorithm using guided local search. GLS explores the area where the new solutions dominate or are of the same rank (non-dominated) with the selected solutions. Therefore, better solutions can be achieved by guided local search. Furthermore, the restriction of selecting only the best results for applying guided local search also helps reduce the computational cost. In each generation, only pre-defined number of the best solutions is selected for applying local search. This provides more diversity towards *Pareto*-optimal solutions

6. 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. [9, 11]), we introduced a guided local search 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 cost due to selection of the best moves. Empirical studies show that the gaps between the obtained results and known lower bounds are small. Furthermore, the results obtained by the MOEA-GLS algorithm almost dominate the results of previous researchers in solving the same benchmarks. It is shown to be capable of obtaining high quality, wide range of non-dominated 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 for integration to the MOEA-GLS algorithm.

7. REFERENCES

[1] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research,* 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," *Mathematics of Operations Research,* vol. 1, pp. 117-129, 1976.

[3] M. Pinedo, *Scheduling theory, algorithms, and systems*: Prentice Hall, second edition, chapter 2, 2002.

[4] Y. Mati and X. Xie, "The complexity of two-job shop problems with multi-purpose unrelated machines," *European Journal of Operational Research,* vol. 152, pp. 159-169, 2004.

[5] J. Carlier and E. Pinson, "An Algorithm for Solving the Job-Shop Problem," *Management Science,* vol. 35, pp. 164-176, Feb. 1989.

[6] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews,* vol. 32, pp. 1-13, 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," *Mathematics and Computers in Simulation,* 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," *European Journal of Operational Research,* vol. 179, pp. 316-333, 2007.

[9] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Computers & Industrial Engineering,* vol. 48, pp. 409-425, 2005.

[10] M. Mastrolilli and L. M. Gambardella, "Effective Neighborhood Functions for the Flexible Job Shop Problem," *Journal of Scheduling,* vol. 3, pp. 3-20, 2000.

[11] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling," *IEEE Transactions on Evolutionary Computation,* vol. 7, pp. 204-223, 2003.

[12] 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, H. W. Krasnogor N., Smith J., Ed.: Springer, pp. 289-312, 2004.

[13] K. Deb, *Multi-objective optimization using evolutionary algorithms*: John Wiley & Sons, 2001.

[14] P. Brucker, B. Jurisch, and A. Krämer, "Complexity of scheduling problems with multi-purpose machines," *Annals of Operations Research,* vol. 70, pp. 57-73, 1997.

[15] 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," *Annals of Discrete Mathematics,* vol. 5, pp. 287-236, 1979.

[16] C. A. C. Coello, "Recent Trends in Evolutionary Multiobjective Optimization," in *Evolutionary Multiobjective Optimization: Theoretical Advances And Applications*, L. J. a. R. G. Ajith Abraham, Ed. London: Springer-Verlag, pp. 7-32, 2005.

[17] J. D. Schaffer, "Multiple Objective Optimization with Vector evaluated Genetic Algorithms," *Proceedings of the First International conference on Genetic Algorithms,* pp. 93-100, 1985.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transaction on Evolutionary Computation,* vol. 6, pp. 181-197, 2002.

[19] E. Zitzler and L. Thiele, "Multi-objective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE transactions on Evolutionary Computation,* vol. 4, pp. 257-271, 1999.

[20] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the Paretor Archived evolution strategy," *Evolutionary Computation,* vol. 8, pp. 149-172, 2000.