Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/100523

Solving The Flexible Job Shop Problem using Multi-Objective Optimizer with Solution Characteristic Extraction

Sheng-Ta Hsieh[1], *Shi-Jim Yen*[2], *Chun-Ling Lin*[3,*], *Shih-Yuan Chiu*[4] *and Tsan-Cheng Su*[2]

[1] Department of Communication Engineering, Oriental Institute of Technology, New Taipei City, Taiwan, R.O.C.
[2] Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan, R.O.C.
[3] Department of Electrical Engineering, Ming Chi University of Technology, New Taipei City, Taiwan, R.O.C.
[4] Systems Development Center, Chung-Shan Institute of Science & Technology, Taoyuan County, Taiwan, R.O.C.

Received: 2 May 2014, Revised: 27 Jan. 2015, Accepted: 17 Feb. 2016
Published online: 1 Sep. 2016

Abstract: It is difficult to find optimal scheduling solutions for abstract scheduling problems with mass parallel tasks on multi-processors because they are NP-complete. In this paper, a solution searching strategy called solution characteristic extraction is proposed as a multi-objective optimizer for solving flexible job shop problems (FJSP). These problems are concerned with finishing assigned jobs with minimal critical machine workload, total workload, and completion times. A suitable job assignment must consider processor performance, job complexity, and job suitability for each individual processor simultaneously. To test the efficiency and robustness of the proposed method, the experiments will contain two groups of benchmarks; with, and without release time constraints. Each benchmark includes numbers of heterogeneous processors and different jobs for execution. The results indicate the proposed method can find more potential solutions, and outperform related methods.

Keywords: Genetic algorithm, job scheduling, multi-objective, optimization, solution characteristic extraction

1 Introduction

In the last decade, the functional requirements of embedded systems have greatly increased. Relying on single processor platforms to meet all expected functions has become unreasonable and unrealistic. In recent times, most advanced embedded systems consist of an implementation of different types of processors. The system architecture is modified to achieve higher productivity, more cost-effectiveness, and more flexibility at implementing applications. Embedded systems today will include different kinds of processor units, such as general microprocessors, digital signal processors (DSPs), and graphics processing units (GPUs), or more, in a single computational platform. Each processing unit is specialized to perform specific functions with high efficiency. Such complex platforms are commonly referred to as heterogeneous platforms [1].

The efficient scheduling of parallel tasks is essential to achieving high performance in multiprocessor systems. However, the complexity of the multiprocessor scheduling problem stems from the existence of many interrelated factors that directly or indirectly contribute to the execution time of the program. To effectively schedule parallel tasks, these competing factors must be considered in the scheduling objective function. Many different scheduling algorithms have been proposed but most consider only a few of these factors simultaneously.

Most current employable techniques of scheduling algorithms are specific to a particular class of problems and work for only a limited range of tasks and system parameters. Little research has been done on employing optimization techniques for dynamic task scheduling, due to the perception that the prohibitive computational cost of these techniques will render them ineffective for dynamic scheduling. The Self-Adjusting Dynamic Scheduling (SADS) class of algorithms utilizes a unified cost model to explicitly account for processor load balance, memory locality, and scheduling overhead at runtime.[2] In [3] and [4], ILP (integer linear programming) is adopted to solve partitioned scheduling

* Corresponding author e-mail: ginnylin@mail.mcut.edu.tw

problems, in which each task is exclusively assigned to a particular processor. Although the studies show that ILP can be used for finding an efficient way for task scheduling, they only consider the computing capacity of each individual processor, which makes the results too vague to accurately express a system module. Furthermore, they fail to mention how much time is spent to find a reasonable solution.

In 2005, Xia *et al*. [5] proposed a method which hybridized PSO and simulated annealing (SA). It involved a variable inertia weight *w* and adopted a weighted concept to transform triple objectives into single objective problems. After the evolution of PSO is complete, SA is employed to adjust operation order. In 2007, Jia *et al*. [6] proposed a method called HPSO to solve FJSP. After the evolution of PSO and a fitness evaluation, the elitism policy of GA keeps healthier particles. Particles with poorer performance were less desirable and regenerated as initializations. The HPSO did not produce significant results and did not attempt to solve large scale problems (such as 10M*15J).

In 2008, Luo *et al*. [7] adopted the Ant Colony method (ACO) to solve FJSP. Its results on the 10M*15J problem was also inconclusive. Ho *et al*. [8] later proposed a method to estimate bounds. In reality, there may exist different schematics that correspond to the optimal fitness value, but Ho's method can find only one. Recently, Zhang *et al*. [9] proposed an interested method *named* NVGA which combined variable neighborhood search (VNS) and GA to solve the multi objective FJSP. It can generate good offspring and keep their diversity during solution searching process. In contrast to the previously mentioned methods, in this paper, a solution searching strategy called *solution characteristic extraction* is proposed to assist a multi-objective genetic algorithm in searching for optimal scheduling solutions. It can significantly improve the solution searching abilities of multi-objective optimizers for solving non-continuous problems. Multiple solutions with the same fitness value can be found, providing various choices.

This paper is organized as follows: Section 2 contains the definition of job scheduling problems. Section 3 briefly introduces the definition of multi-objective optimization. Section 4 introduces the genetic algorithm. Section 5 presents the proposed methods. Section 6 includes the experiment results and compares the proposed method to related works. Section 7 of the paper contains the conclusion.

2 Problem Definition

The Flexible Job Shop Problem (FJSP) is an extension of the classical job shop scheduling problem which allows an operation to be processed by any machine from a given set. The problem is to assign each operation to a machine, order the operations on the machines, and ensure the maximum completion time (Makespan), critical machine workload ($\mathrm{Max}(W_k)$) and total workload (W_{td}) of all operations are minimized [10]. Scheduling strategies can be classified as either static or dynamic, depending on when the task assignment decisions are made. Static scheduling is also referred to as pre-scheduling and determines on compilation-time which tasks should be executed by which processors. Dynamic scheduling strategies move the locus of control from the compiler to one or more processors that distribute work to themselves and/or to other processors as needed at runtime to balance the system load.

Jobs / Machines	$O_{1.1}$	$O_{1.2}$	$O_{1.3}$	$O_{2.1}$	$O_{2.2}$	$O_{3.1}$
M_1	1	2	2	2	2	5
M_2	2	2	1	3	3	4
M_3	1	3	3	4	4	3

Fig. 1: An example of flexible job shop problem

Time(s) / Machines	1	2	3	4	5	6	7
M_1	$O_{2.1}$				$O_{1.2}$		
M_2			$O_{2.2}$				$O_{1.3}$
M_3	$O_{3.1}$			$O_{1.1}$			

Fig. 2: A schematic example of job assignment

A perfectly balanced load does not imply that all processors must execute the same number of parallel tasks since tasks typically have variable execution times. Hence, it is not enough to simply divide the total number of tasks evenly amongst each processor. A well scheduled strategy will increase system performance, fully utilize system computation ability and reduce total execution time and reduce power consumption.

An example of FJSP is illustrated in Fig. 1. In the figure, there are three Jobs that require execution and three machines to complete them. Each Job may contain several operations. Job 1 can be separated into $O_{1.1}$, $O_{1.2}$ and $O_{1.3}$; Job 2 can be separated into $O_{2.1}$ and $O_{2.2}$, and Job 3 contains $O_{3.1}$. The required completion times of the operations as executed by each machine are represented in the table. For example, it takes machine 1 (M_1) 5 seconds to execute $O_{3.1}$.

The goal is to finish all jobs with minimal critical machine workload ($\mathrm{Max}(W_k)$), total workload (W_{td}) and completion times (Makespan). After Job assignment, each operation will be appropriately distributed and executed by the assigned machine. The schematics of Job assignment is illustrated in Fig. 2.

Thus, $O_{1.1}$ is assigned to M_3; $O_{1.2}$ is assigned to M_1; $O_{1.3}$ is assigned to M_2; $O_{2.1}$ is assigned to M_1 and $O_{2.2}$ is assigned to M_2; and $O_{3.1}$ is assigned to M_3. It can be seen

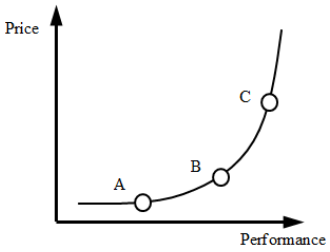

Fig. 3: An example of MOP - buying a processor

in Fig. 2 that the Max(W_k) is 4 seconds; the W_{td} is 12 seconds; and the Makespan is 7 seconds. The solution in the example above may not be the optimal one; there may exist more potential (better) solutions. It is also possible in FJSP for different solutions to have corresponding fitness values. Therefore, a capable optimizer is needed to explore the solution space to find potential solutions.

3 Multi-Objective Optimization

The main difference between single-objective problems (SOPs) and multi-objective problems (MOPs) is MOPs contain more than one objective that needs to be accomplished simultaneously. Such problems are encountered in many applications, where two or more, sometimes competing and/or incommensurable objective functions have to be minimized concurrently. In other words, problems must arrive at a single optimal solution which satisfies all objectives simultaneously. Even if a problem contains more than one objective to be solved, it is still considered a SOP. A MOP must contain competing objectives; i.e. the objectives cannot be optimized simultaneously. For example, suppose costumers want to buy a processor with a lower cost and higher performance for their computer. The two objectives (low cost and high performance) are conflicting. The choices of processors are illustrated in Fig. 3. The three options A, B, and C marked in Fig. 3 are all non-dominated optimal solutions. These optimal solution sets can constitute a continuous or non-continuous curve. This curve is called the Pareto front. All solutions located on it are considered optimal solutions. In other words, unlike SOPs, which have only one optimal solution, MOPs have a set (more than one) of optimal solutions.

Due to the multi-criteria nature of MO problems, the "optimality" of a solution has to be redefined, giving rise to the concept of Pareto optimality. In contrast to the single–objective optimization case, MO problems are characterized by trade–offs and thus, a multitude of Pareto optimal solutions [11].

4 Genetic Algorithm

The genetic algorithm (GA) is used to evolve one population of chromosomes into a new population by employing a principle similar to Darwin's "natural selection"; together with the genetics-inspired operators of selection, cross-over, mutation, and inversion. The basic principles of GA were first introduced by John Holland in 1975[12]-[13]. Holland's GA was the first evolutionary computation (EC) paradigm developed and applied.

In genetic algorithms, the fitter chromosome is more likely to be selected for reproduction. The objective of cross-over is to choose random loci and exchange the subparts of chromosomes to create offspring. Mutation randomly flips the allele values of some locations in the chromosome; and inversion reverses the order of a contiguous section of the chromosome [14]. The term chromosome typically refers to a candidate solution to a problem. Through the progress of genetic evolution, genetic algorithms can find solutions efficiently without derivative information; an optimal solution will be represented by a final chromosome winning the genetic competition.

The traditional GA (TGA) has the following features: (1) a bit string representation; (2) proportional selection; (3) cross-over as the primary method to produce new individuals; (4) mutation for disturbing evolution to prevent solutions being bounded to local search; and (5) the application of elitism policies. In this section, a brief introduction of genetic algorithm will be described.

4.1 Chromosome Representation

Consider that a problem is presented as $f(x_1, x_2, x_3,..., x_N)$,which consists of N tunable parameters to be optimized. In GA, it can be encoded in vector representation (i.e., chromosome) $C^m=(x_1, x_2, x_3,..., x_N)$, m=1, 2, ..., p,where p denotes the population size. For high-dimension or complex problems, GA will require a larger population for a uniform distribution of population in the search space; otherwise, it may be unable to discover all possible solutions. The value p is always given experimentally. Thus, the chromosomes representation is shown in Fig. 4.

4.2 Initial Population

For most optimal techniques, the final solutions are usually restricted by the initialization. However, GA is able to overcome this drawback with the cross-over and mutation operation. Therefore, chromosomes can be scattered in an area of first generation. The initial population will be used to generate p chromosomes which will be distributed over the searching space uniformly.

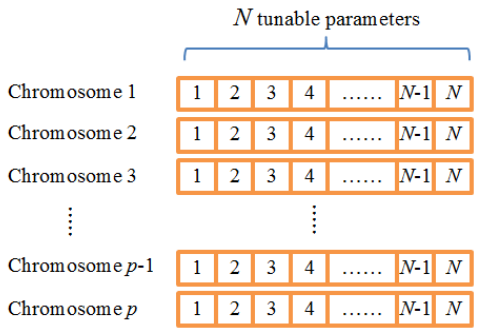

Fig. 4: Chromosome representation

4.3 Cross-over

The cross-over operation produces new chromosomes (offspring) by selecting two random parent chromosomes, but it doesn't guarantee that all offspring are better than the parents. However, after adopting "exploration" and "exploitation" during the performance of the cross-over [15], good results can be ensured because the offspring will be generated around better parents. The details of "exploration" and "exploitation" are described in next section. The number of individuals joined during cross-over is based on a pre-defined parameter r_c which is called cross-over rate. Thus, there will be $round(p \times r_c)$ individuals (parents) joined during cross-over.

4.4 Mutation

The mutation operator randomly changes some subparts of a chromosome. When GA is adapting, the chromosomes will move to the closest optimal solution to itself, but that may not be a global optimization. Therefore, disturbances to extend the search range are quite important. In general, the offspring of mutation O_m is generated inside the search space, randomly as $O_m = \beta$, where β denotes a mutation vector with random components of uniform distribution in the searching space. The number of parents which joins mutation is based on a predefined parameter r_m which is called mutation rate. Thus, there are mutation $round(p \times r_m)$ individuals (parents) that will be joined during mutation.

In general, the fitness of mutated offspring may randomly be better or worse than their parents and/or any cross-over offspring. On the other hand, adopting the mutation operation will extend the search range to survey unexplored areas in the search space, and find potential optimal solutions.

4.5 Selection

After cross-over and mutation operations, the number of all chromosomes, including parents and offspring in a population, will be larger than the number during initialization. In order to produce better offspring, the elitism operation is adapted to select p better chromosomes to survive the next generation.

GA optimization is combined with operations mentioned above, and repeats the evolution process until it reaches the pre-defined terminating conditions.

5 The Proposed Method

Because the solutions of job shop problems are non-continued, such MO problems may have a mass of solutions which correspond to the same fitness value. Preserving these solutions can provide additional optimal selections. The solution characteristic extraction is proposed for enhancing the solution searching ability for non-continued MO problems, especially for flexible job-shop scheduling problems.

It is important for offspring generated by cross-over or mutation during multi-objective genetic algorithm to offer more potential information and diversity. Offspring with higher potential information and diversity (which equals more available optimal solutions), will be found and then kept in each generation while an offspring with lower potential will decrease GA's solution searching ability. An improved GA is proposed to increase the solution searching capabilities of MOGA, for solving non-continuous MO problems.

Table 1: An example for probability table

Machines	Completion Time	Completion Time	Probability
$\boldsymbol{M}_1$	1	1	0.55
$\boldsymbol{M}_2$	3	0.33	0.18
$\boldsymbol{M}_3$	2	0.5	0.27

5.1 Initialization

First, a probability table is generated for the distribution of each operation because an operation will have different completion times when executed by different machines. The probability table of an operation is generated according to their completion time (reciprocal). A machine with shorter completion time for executing an operation will have a higher probability of being chosen for the task. An example of probability table is listed in Table 1. The probability of M_1, M_2, and M_3 are calculated

Jobs	Operations		
J_1	$O_{1.1}$	$O_{1.2}$	$O_{1.3}$
J_2	$O_{2.1}$	$O_{2.2}$	
J_3	$O_{3.1}$		

Fig. 5: An example of jobs' content

by 1/(1+0.33+0.5), 0.33/(1+0.33+0.5), and 0.5/(1+0.33+0.5) respectively.

An example of the contents of the jobs is illustrated in Fig. 5. Job 1(J_1), job 2(J_2), and job 3(J_3) can be separated into 3 operations, 2 operations, and 1 operation respectively. In other words, J_1 consists of operations 1.1($O_{1.1}$), 1.2($O_{1.2}$) and 1.3($O_{1.3}$); J_2 consists of operations 2.1($O_{2.1}$) and 2.2($O_{2.2}$); J_3 consists of operation 3.1($O_{3.1}$). Furthermore, operations belonging to the same job must follow an order of execution. For example, operation 1.1 must be finished before the execution of operation 1.2, and operation 1.2 must be finished before the execution of operation 1.3, and so on.

Assume there are 3 machines to execute the three jobs mentioned above. The operation assignment and their corresponding completion times are shown in Fig. 1. Different machines may perform at different efficiencies when executing each operation; from this table, for example, M_1 (machine 1) needs 5 seconds to finish $O_{3.1}$ but M_3 needs only 3 seconds. Thus, the probability table of $O_{3.1}$ is shown as Table 2.

Table 2: An example probability table for $O_{3.1}$

Machines	Completion Time	$\frac{1}{\text{Completion Time}}$	Probability
M_1	5	0.2	0.25
M_2	4	0.25	0.32
M_3	3	0.33	0.43

An initialization example for FJSP is listed in Table 3. First row represents operation number and second row represents operation assignment information. The information consists of two parts, the integer, and the float. The integer, which is generated by the probability table, represents the machine to be assigned. And the float, which is a random number between 0 and 1, represents operations' order number.

In Table 3, the number 3.65 corresponds to operation 1.1 ($O_{1.1}$). This means operation 1.1($O_{1.1}$) is assigned to machine 3 (M_3) and its order number is 65. An operation with a bigger order number has a higher priority to be assigned to a machine. The consecutive relation between each operation is also taken into consideration. In Table 3,

Table 3: An initialization example for FJSP

Operations	$O_{1.1}$	$O_{1.2}$	$O_{1.3}$	$O_{2.1}$	$O_{2.2}$	$O_{3.1}$
Assigned Information	3.65	1.18	2.39	1.77	2.57	3.26

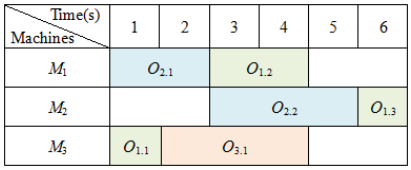

Fig. 6: The operations assignment and their execution time-line

$O_{2.1}$ is assigned to machine 1 (M_1) and $O_{1.1}$ is assigned to machine 3 (M_3). Later, $O_{2.2}$ is assigned to machine 2 (M_2) and $O_{3.1}$ is assigned to machine 3 (M_3). Finally, $O_{1.2}$ is assigned to machine 1 (M_1) and $O_{1.3}$ is assigned to machine 2 (M_2). In other words, the executing sequence can be rewritten as follows:

1. $O_{2.1}$ is assigned to M_1: because $max(J_1, J_2, J_3) = max(O_{1.1}, O_{2.1}, O_{3.1}) = max(65, 77, 26) = 77$.
2. $O_{1.1}$ is assigned to M_3: because $max(J_1, J_2, J_3) = max(O_{1.1}, O_{2.2}, O_{3.1}) = max(65, 57, 26) = 65$.
3. $O_{2.2}$ is assigned to M_2: because $max(J_1, J_2, J_3) = max(O_{1.2}, O_{2.2}, O_{3.1}) = max(18, 57, 26) = 57$.
4. $O_{3.1}$ is assigned to M_3: because $max(J_1, J_3) = max(O_{1.2}, O_{3.1}) = max(18, 26) = 26$ (J_2 complete).
5. $O_{1.2}$ is assigned to M_1: because $max(J_1) = max(O_{1.2}) = max(18) = 18$ (J_3 completion).
6. $O_{1.3}$ is assigned to M_2: because $max(J_1) = max(O_{1.3}) = max(39) = 39$ (All Jobs are complete).

These assigned operations and their execution time-lines are exhibited in Fig. 6.

There are three steps to transform a system job scheduling into a schematic.

1. Estimate completion time of previous operations which belong to the same job.
2. Estimate the available time for the machine to join the next unexecuted operation.
3. The maximum time of the two items above is the timing to join subsequent operations.

An operation's execution and assignment must take into account both the order of execution and rationality of execution by a machine. The proposed method ensures that once an operation is finished, a subsequent operation can be executed immediately by an idle machine. For example, according to the order of operations, M_3will finish the execution of $O_{1.1}$ by the 2^{nd} second; thus $O_{1.2}$ can be executed since the 2^{rd} second. On the other hand, taking into account the rationality of execution by a machine, because M_1 is occupied processing $O_{2.1}$ during

the first 2 seconds, the machine is available only after the 3^{rd} second. The maximum time between the next operation's availability and the machine's availability is 3 seconds (max(2, 3) = 3); therefore, $O_{1.2}$ can be executed since the 3^{rd} second.

It seems that the proposed initialization step is a little bit complex. But, it comes with the advantage of ensuring that each combination of job assignment can be reversed because different codes may correspond to the same schematic. The chromosome initialization is based on a probability table to assign each job to a corresponding machine. This is the "integer" of the proposed coding method mentioned above. A randomly generated fraction is then added to each integer, as its order number.

5.2 Cross-over

A population in GA consists of a number of chromosomes. Each chromosome represents a potential solution of the optimization task. All of the chromosomes iteratively discover a probable solution. In the cross-over operation, each chromosome will be generated based on its parents. This combines the information from two chromosomes that were also generated by chromosomes in previous generations and evaluated by the cost function; finally, better (fitter) chromosomes are kept in the population. If a chromosome discovers a new probable solution, its offspring will move closer to it to explore the region in more detail during the cross-over process.

In the proposed method, the real-coded genetic algorithm is adopted to perform cross-over between chromosomes. For example, assume that two chromosomes, C_1, and C_2 are randomly picked from a population for cross-over, and C_1 is better than C_2. The offspring O_C can be obtained by exploration cross-over using:

$$O_C = C_1 + \alpha (C_1 - C_2) \quad (1)$$

where α is a random value between [0, 1]. On the other hand, the offspring O_C can also be obtained by exploitation cross-over using:

$$O_C = C_1 - \alpha (C_1 - C_2) \quad (2)$$

In the proposed method, both the exploration and the exploitation methods are adopted. Interpolation will restrict the search range to a smaller area, but extrapolation will extend the search range. The exploration and exploitation cross-over strategies are showed in Fig. 7.

In traditional GA, cross-over is performed to discover additional potential solutions. In machine assignment of FJSP, such solution space is non-continued; the traditional cross-over approach is not of much use at finding solutions. Thus, in the proposed method, the cross-over for the integer-part of the chromosome (for machine assignment) is performed by integer-exchange, and the float-part of the chromosome (for executing sequence) is performed by traditional cross-over.

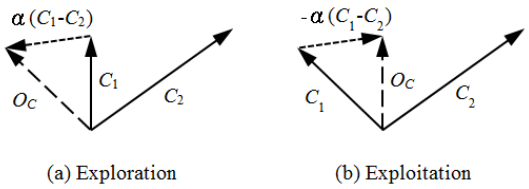

Fig. 7: Exploration and exploitation strategies of cross-over

5.3 Mutation

In general, mutation is adopted to generate new chromosomes, mutating one or more genes, to salvage chromosomes that have been trapped in the local minimum through random process, and to discover other potential searching spaces. It will allow additional potential solutions to be produced during solution exploration, and explore unsearched solution space.

Mutation in the proposed method is performed at initialization to selected chromosomes. The chosen chromosomes are assigned to a new corresponding machine (integer part) according to the probability table and also given a randomly generated order number (fraction part).

Table 4: Two simply job assignment

Operations	$O_{1.1}$	$O_{1.2}$	$O_{2.1}$	$O_{2.2}$	$O_{3.1}$
Assigned Probabilities	3.65	1.72	2.58	4.99	4.80

(a)

Operations	$O_{1.1}$	$O_{1.2}$	$O_{2.1}$	$O_{2.2}$	$O_{3.1}$
Assigned Probabilities	3.15	1.23	2.04	4.50	4.39

(b)

5.4 Normalization

In order to prevent the evolution process from driving chromosomes toward unreasonable solution spaces, after cross-over and mutation all contents in chromosomes need to be normalized. For example, there are two simple job assignments presented in Table 4. Each table presents a decoded result from a chromosome.

Time(s) / Machines	1	2	3	4
M_1	$O_{2.1}$		$O_{1.2}$	$O_{3.2}$
M_2	$O_{1.1}$		$O_{2.2}$	
M_3	$O_{3.1}$			$O_{1.3}$

Fig. 8: An example of job assignment schematic

2.1	2.1	1.2	3.2
1.1	0	2.2	2.2
3.1	3.1	0	1.3

Fig. 9: An re-organized schematic of job assignment

$e_{1,1}$	$e_{1,2}$	$e_{1,3}$	$e_{1,4}$
$e_{2,1}$	$e_{2,2}$	$e_{2,3}$	$e_{2,4}$
$e_{3,1}$	$e_{3,2}$	$e_{3,3}$	$e_{3,4}$

Fig. 10: Using symbols to replace elements in Fig. 9

$E_{1,1} = 1 \times e_{1,1}$	$E_{1,2} = 2 \times e_{1,2}$	$E_{1,3} = 3 \times e_{1,3}$	$E_{1,4} = 4 \times e_{1,4}$
$E_{2,1} = 5 \times e_{3,1}$	$E_{2,2} = 6 \times e_{2,2}$	$E_{2,3} = 7 \times e_{2,3}$	$E_{2,4} = 8 \times e_{2,4}$

Fig. 11: The weighted elements

It seems the contents are different between Table 4 (a) and (b). The two represent the same job assignment and executing sequence from two different chromosomes. If two chromosomes are selected for cross-over, the fraction part of the offspring will increase if they are generated by exploration, and decrease if generated by exploitation. This will influence the evolution process towards the right direction. In some cases, the solution searching may stand still. Normalization in the proposed method can solve this issue; the details of which are as follows.

1. Because each chromosome has 5 genes, the fraction part is equally divided into 5 numbers. They are 0.0, 0.2, 0.4, 0.6 and 0.8.
2. All genes are sorted according to their fraction part. The genes with the smallest fraction have it replaced by 0.0, the genes with the median fraction have it replaced by 0.4, and genes with the largest fraction have it replaced by 0.8, and so on.

After normalization, the genes from Table 4 (a) and (b) can be treated as the same genes, as shown in Table 5.

Table 5: The normalized genes

Operations	$O_{1.1}$	$O_{1.2}$	$O_{2.1}$	$O_{2.2}$	$O_{3.1}$
Assigned Probabilities	3.20	1.40	2.00	4.80	4.60

This process reduces situations where different chromosome contents correspond to the same job assignment, and streamlines the evolution process.

5.5 Solution Characteristic Extraction

After cross-over and mutation, all job assignments can be stored as a two dimensional matrix. Because there is an amassing of solutions reserved in the external repository, the computational consumption is also increasing. The solution characteristic extraction is proposed to lighten computational consumption during matrix comparison.

For example, there are 3 jobs, which consist of several operations, and need to be executed by 3 available machines. The job assignment schematic is expressed in Fig. 8. Each operation will be fitted into a time slot by job assignment. In order to assign each time slot with a unique weight, the schematic can be re-organized as expressed in Fig. 9. The natural number of operations is labeled on time slots and empty time slots are assigned as 0.

Further explanation of the weight calculation can be found in Fig. 10, which replaces Fig. 9. The elements (natural number) are replaced with symbols. For example, *A* replaced 2.1, *E* replaced 2.1 and *K* replaced 0, etc.

Then, each symbol in Fig. 10 is multiplied by its own weight value according to its location on the matrix. The weighted elements are listed in Fig. 11.

After that, the two-dimensional matrix is then transformed to one-dimensional, which is listed in Fig. 12.

Each element of this new matrix is further multiplied by its own weighted value. It is also according to its location of the table. The further weighted elements are listed in Fig. 13.

Finally, the proposed characteristic extraction is applied to find the Euclid distance of this weighted matrix to find its characteristic-value. Let Q denotes the characteristic-value, the Q value is calculated by

$$Q = \left(W_1^2 + W_2^2 + W_3^2 + W_4^2 + W_5^2 + W_6^2 + W_7^2\right)^{0.5} \quad (3)$$

The proposed characteristic extraction can rapidly compare two job assignments regardless if whether they are different or the same.

$D_1 = E_{1,1} + E_{1,2} + E_{1,3} + E_{1,4}$	$D_2 = E_{2,1} + E_{2,2} + E_{2,3} + E_{2,4}$	$D_3 = E_{3,1} + E_{3,2} + E_{3,3} + E_{3,4}$	$D_4 = E_{1,1} + E_{2,1} + E_{3,1}$	$D_5 = E_{1,2} + E_{2,2} + E_{3,2}$	$D_6 = E_{1,3} + E_{2,3} + E_{3,3}$	$D_7 = E_{1,4} + E_{2,4} + E_{3,4}$

Fig. 12: The transformed one-dimensional matrix

$W_1 = 1 \times D_1$	$W_2 = 2 \times D_2$	$W_3 = 3 \times D_3$	$W_4 = 4 \times D_4$	$W_5 = 5 \times D_5$	$W_6 = 6 \times D_6$	$W_7 = 7 \times D_7$

Fig. 13: The further weighted elements

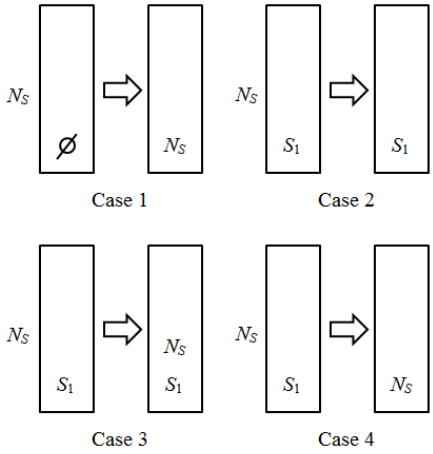

Fig. 14: Possible case for archive controller

5.6 External Repository

An external repository is utilized in the proposed method in order to reserve non-dominated solutions which are of the same fitness value but correspond to unique job assignments. The function of the repository controller is to make a decision for whether or not certain solutions should be included in the archive. After completing the evolutionary process in the current generation, all new solutions are compared to previous results. New solutions which have the same characteristic-value as the solutions reserved in the external repository will be eliminated. After a quick characteristic-value comparison, the remaining new solutions will be further assessed in before being deposited in the external repository. The decision-making process is as follows.

1. If the external archive is empty, the non-dominated solutions found currently will always be accepted (Case 1, in Fig. 14).
2. If the new solution is dominated by any individual in the external repository, then such a solution will be discarded (Case 2, in Fig. 14).
3. Otherwise, if none of the solutions contained in the external repository dominates the new solution, then such a solution will be stored in the external repository. If there are solutions in the external repository that are dominated by the new solution, then such dominated solutions will be removed from the external repository (Case 3 and 4, in Fig. 14).

Table 6: Problem 8 X 8 with 27 operations (partial flexibility)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J_1	$O_{1.1}$	5	3	5	3	3	X	10	9
	$O_{1.2}$	10	X	5	8	3	9	9	6
	$O_{1.3}$	X	10	X	5	6	2	4	3
J_2	$O_{2.1}$	5	7	3	9	8	X	9	X
	$O_{2.2}$	X	8	5	2	6	7	10	9
	$O_{2.3}$	X	10	X	5	6	4	1	7
	$O_{2.4}$	10	8	9	6	4	7	X	X
J_3	$O_{3.1}$	10	X	X	7	6	5	2	4
	$O_{3.2}$	X	10	6	4	8	9	10	X
	$O_{3.3}$	1	4	5	6	X	10	X	7
J_4	$O_{4.1}$	3	1	6	5	9	7	8	4
	$O_{4.2}$	12	11	7	8	10	5	6	9
	$O_{4.3}$	4	6	2	10	3	9	5	7
J_5	$O_{5.1}$	3	6	7	8	9	X	10	X
	$O_{5.2}$	10	X	7	4	9	8	6	X
	$O_{5.3}$	X	9	8	7	4	2	7	X
	$O_{5.4}$	11	9	X	6	7	5	3	6
J_6	$O_{6.1}$	6	7	1	4	6	9	X	10
	$O_{6.2}$	11	X	9	9	9	7	6	4
	$O_{6.3}$	10	5	9	10	11	X	10	X
J_7	$O_{7.1}$	5	4	2	6	7	X	10	X
	$O_{7.2}$	X	9	X	9	11	9	10	5
	$O_{7.3}$	X	8	9	3	8	6	X	10
J_8	$O_{8.1}$	2	8	5	9	X	4	X	10
	$O_{8.2}$	7	4	7	8	9	X	10	X
	$O_{8.3}$	9	9	X	8	5	6	7	1
	$O_{8.4}$	9	X	3	7	1	5	8	X

Table 7: Problem 10 X 10 with 30 operations (total flexibility)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	$O_{1.1}$	1	4	6	9	3	5	2	8	9	5
	$O_{1.2}$	4	1	1	3	4	8	10	4	11	4
	$O_{1.3}$	3	2	5	1	5	6	9	5	10	3
J_2	$O_{2.1}$	2	10	4	5	9	8	4	15	8	4
	$O_{2.2}$	4	8	7	1	9	6	1	10	7	1
	$O_{2.3}$	6	11	2	7	5	3	5	14	9	2
J_3	$O_{3.1}$	8	5	8	9	4	3	5	3	8	1
	$O_{3.2}$	9	3	6	1	2	6	4	1	7	2
	$O_{3.3}$	7	1	8	5	4	9	1	2	3	4
J_4	$O_{4.1}$	5	10	6	4	9	5	1	7	1	6
	$O_{4.2}$	4	2	3	8	7	4	6	9	8	4
	$O_{4.3}$	7	3	12	1	6	5	8	3	5	2
J_5	$O_{5.1}$	7	10	4	5	6	3	5	15	2	6
	$O_{5.2}$	5	6	3	9	8	2	8	6	1	7
	$O_{5.3}$	6	1	4	1	10	4	3	11	13	9
J_6	$O_{6.1}$	8	9	10	8	4	2	7	8	2	10
	$O_{6.2}$	7	3	12	5	4	3	6	9	2	15
	$O_{6.3}$	4	7	3	6	3	4	1	5	1	11
J_7	$O_{7.1}$	1	7	8	3	4	9	4	13	10	7
	$O_{7.2}$	3	8	1	2	3	6	11	2	13	3
	$O_{7.3}$	5	4	2	1	2	1	8	14	5	7
J_8	$O_{8.1}$	5	7	11	3	2	9	8	5	12	8
	$O_{8.2}$	8	3	10	7	5	13	4	6	8	4
	$O_{8.3}$	6	2	13	5	4	3	5	7	9	5
J_9	$O_{9.1}$	3	9	1	3	8	1	6	7	5	4
	$O_{9.2}$	4	6	2	5	7	3	1	9	6	7
	$O_{9.3}$	8	5	4	8	6	1	2	3	10	12
J_{10}	$O_{10.1}$	4	3	1	6	7	1	2	6	20	6
	$O_{10.2}$	3	1	8	1	9	4	1	4	17	15
	$O_{10.3}$	9	2	4	2	3	5	2	4	10	23

5.7 Algorithm Flow of Proposed Method

The flowchart of the proposed method is shown in Fig. 15. After initialization new chromosomes (offspring) will be generated by either cross-over or mutation of MOGA to generate probability tables for each operation. Both "exploration" and "exploitation" strategies of cross-over are adopted for both widespread and comprehensive searches of the solution space. Cross-over will randomly choose two chromosomes (parents) to exchange their information and try to generate a better chromosome. Mutation is also introduced in preventing chromosomes from falling into local optimal and pointing them to explore unsearched solution space. All contents of chromosomes, which are generated by cross-over and mutation, will then be normalized. The fitness values are then calculated and better chromosomes are kept. Each chromosome will represent a unique job assignment case; their characteristics will be extracted to rapidly compare them with the external repository. All non-dominated solutions are then reserved in external repository.

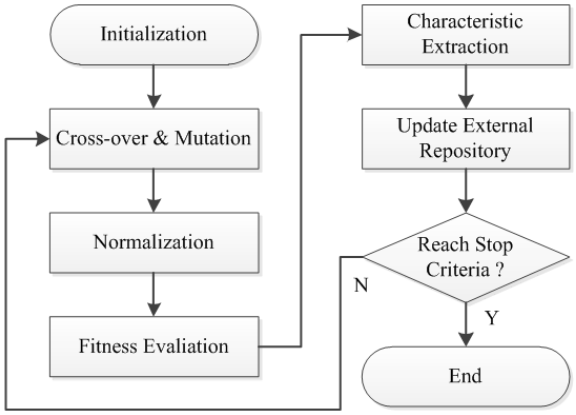

Fig. 15: The flowchart of proposed method

Table 8: Problem 15 X 10 with 56 operations (total flexibility

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	$O_{1.1}$	1	4	6	9	3	5	2	8	9	4
	$O_{1.2}$	1	1	3	4	8	10	4	11	4	3
	$O_{1.3}$	2	5	1	5	6	9	5	10	3	2
	$O_{1.4}$	10	4	5	9	8	4	15	8	4	4
J_2	$O_{2.1}$	4	8	7	1	9	6	1	10	7	1
	$O_{2.2}$	6	11	2	7	5	3	5	14	9	2
	$O_{2.3}$	8	5	8	9	4	3	5	3	8	1
	$O_{2.4}$	9	3	6	1	2	6	4	1	7	2
J_3	$O_{3.1}$	7	1	8	5	4	9	1	2	3	4
	$O_{3.2}$	5	10	6	4	9	5	1	7	1	6
	$O_{3.3}$	4	2	3	8	7	4	6	9	8	4
	$O_{3.4}$	7	3	12	1	6	5	8	3	5	2
J_4	$O_{4.1}$	6	2	5	4	1	2	3	6	5	4
	$O_{4.2}$	8	5	7	4	1	2	36	5	8	5
	$O_{4.3}$	9	6	2	4	5	1	3	6	5	2
	$O_{4.4}$	11	4	5	6	2	7	5	4	2	1
J_5	$O_{5.1}$	6	9	2	3	5	8	7	4	1	2
	$O_{5.2}$	5	4	6	3	5	2	28	7	4	5
	$O_{5.3}$	6	2	4	3	6	5	2	4	7	9
	$O_{5.4}$	6	5	4	2	3	2	5	4	7	5
J_6	$O_{6.1}$	4	1	3	2	6	9	8	5	4	2
	$O_{6.2}$	1	3	6	5	4	7	5	4	6	5
J_7	$O_{7.1}$	1	4	2	5	3	6	9	8	5	4
	$O_{7.2}$	2	1	4	5	2	3	5	4	2	5
J_8	$O_{8.1}$	2	3	6	2	5	4	1	5	8	7
	$O_{8.2}$	4	5	6	2	3	5	4	1	2	5
	$O_{8.3}$	3	5	4	2	5	49	8	5	4	5
	$O_{8.4}$	1	2	36	5	2	3	6	4	11	2
J_9	$O_{9.1}$	6	3	2	22	44	11	10	23	5	1
	$O_{9.2}$	2	3	2	12	15	10	12	14	18	16
	$O_{9.3}$	20	17	12	5	9	6	4	7	5	6
	$O_{9.4}$	9	8	7	4	5	8	7	4	56	2
J_{10}	$O_{10.1}$	5	8	7	4	56	3	2	5	4	1
	$O_{10.2}$	2	5	6	9	8	5	4	2	5	4
	$O_{10.3}$	6	3	2	5	4	7	4	5	2	1
	$O_{10.4}$	3	2	5	6	5	8	7	4	5	2
J_{11}	$O_{11.1}$	1	2	3	6	5	2	1	4	2	1
	$O_{11.2}$	2	3	6	3	2	1	4	10	12	1
	$O_{11.3}$	3	6	2	5	8	4	6	3	2	5
	$O_{11.4}$	4	1	45	6	2	4	1	25	2	4
J_{12}	$O_{12.1}$	9	8	5	6	3	6	5	2	4	2
	$O_{12.2}$	5	8	9	5	4	75	63	6	5	21
	$O_{12.3}$	12	5	4	6	3	2	5	4	2	5
	$O_{12.4}$	8	7	9	5	6	3	2	5	8	4
J_{13}	$O_{13.1}$	4	2	5	6	8	5	6	4	6	2
	$O_{13.2}$	3	5	4	7	5	8	6	6	3	2
	$O_{13.3}$	5	4	5	8	5	4	6	5	4	2
	$O_{13.4}$	3	2	5	6	5	4	8	5	6	4
J_{14}	$O_{14.1}$	2	3	5	4	6	5	4	85	4	5
	$O_{14.2}$	6	2	4	5	8	6	5	4	2	6
	$O_{14.3}$	3	25	4	8	5	6	3	2	5	4
	$O_{14.4}$	8	5	6	4	2	3	6	8	5	4
J_{15}	$O_{15.1}$	2	5	6	8	5	6	3	2	5	4
	$O_{15.2}$	5	6	2	5	4	2	5	3	2	5
	$O_{15.3}$	4	5	2	3	5	2	8	4	7	5
	$O_{15.4}$	6	2	11	14	2	3	6	5	4	8

Table 9: Problem 4 X 5 with 27 operations (partial flexibility)

		M_1	M_2	M_3	M_4	M_5
J_1	$O_{1.1}$	2	5	4	1	2
	$O_{1.2}$	5	4	5	7	5
	$O_{1.3}$	4	5	5	4	5
J_2	$O_{2.1}$	2	5	4	7	8
	$O_{2.2}$	5	6	9	8	5
	$O_{2.3}$	4	5	4	54	5
J_3	$O_{3.1}$	9	8	6	7	9
	$O_{3.2}$	6	1	2	5	4
	$O_{3.3}$	2	5	4	2	4
	$O_{3.4}$	4	5	2	1	5
J_4	$O_{4.1}$	1	5	2	4	12
	$O_{4.2}$	5	1	2	1	2

Table 10: Problem 10 X 7 with 30 operations (total flexibility)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7
J_1	$O_{1.1}$	1	4	6	9	3	5	2
	$O_{1.2}$	8	9	5	4	1	1	3
	$O_{1.3}$	4	8	10	4	11	4	3
J_2	$O_{2.1}$	6	9	8	6	5	10	3
	$O_{2.2}$	2	10	4	5	9	8	4
	$O_{2.3}$	15	4	8	4	8	7	1
J_3	$O_{3.1}$	9	6	1	10	7	1	6
	$O_{3.2}$	11	2	7	5	2	3	14
	$O_{3.3}$	2	8	5	8	9	4	3
J_4	$O_{4.1}$	5	3	8	1	9	3	6
	$O_{4.2}$	1	2	6	4	1	7	2
	$O_{4.3}$	7	1	8	5	4	3	9
J_5	$O_{5.1}$	2	4	5	10	6	4	9
	$O_{5.2}$	5	1	7	1	6	6	2
	$O_{5.3}$	8	7	4	56	9	8	4
J_6	$O_{6.1}$	5	14	1	9	6	5	4
	$O_{6.2}$	3	5	2	5	4	5	7
	$O_{6.3}$	1	4	6	9	3	5	2
J_7	$O_{7.1}$	5	6	3	6	5	15	2
	$O_{7.2}$	6	5	4	9	5	4	3
	$O_{7.3}$	9	8	2	8	6	1	7
J_8	$O_{8.1}$	6	1	4	1	10	4	3
	$O_{8.2}$	11	13	9	8	9	10	8
	$O_{8.3}$	4	2	7	8	3	10	7
J_9	$O_{9.1}$	12	5	4	5	4	5	5
	$O_{9.2}$	4	2	15	99	4	7	3
	$O_{9.3}$	9	5	11	2	5	4	2
J_{10}	$O_{10.1}$	9	4	13	10	7	6	8
	$O_{10.2}$	4	3	25	3	8	1	2
	$O_{10.3}$	1	2	6	11	13	3	5

6 Experiments

6.1 Benchmarks of Job Shop Problems

In the experiments, two groups of benchmarks (G-1 and G-2) are included. The main difference between the two groups of benchmarks is that G-1 does not put into consideration release time. It includes three sets of benchmarks, which are 8 x 8, 10 x 10 and 15 x 10, and are listed in Table 6 to 8. Note that the 15 x 10 problem indicates that there are 10 available processors to perform 15 Jobs. The "X" marks present that some operations cannot be executed on specific machines. For example, in table 6, the $O_{1.1}$ cannot be executed on M_6. Thus, this operation ($O_{1.1}$) can be assigned to any machines except M_6. The G-2 does take release time into consideration. It also includes three sets of benchmarks, which are 4 x 5, 10 x 7 and 15 x 10, and are listed in Table 9, 10 and 8. Because both G-1 and G-2 contain 15 x 10 problems, they are listed under the same table (Table 8). The release time is an executing constraint which limits jobs in a benchmark to release after a few seconds. For example, a job release time is assigned for $r2 = 3$. It means job 2 will be released after three seconds. In other words, job 2 only becomes available for execution after the third second. The release times of G-2 are listed in Table 11.

Table 11: Release time of G-2

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

6.2 Experiment Results

In the experiments, two groups of benchmarks are included to compare the performance of the proposed method with other related works, such as MOEA-GLS [8], hybrid PSO & SA [5] and VNGA [9]. The parameters used in the two comparative methods are defined according to their original settings. For the proposed method, population size is set as 1000, cross-over rate is set as 0.95 and mutation rate is 0.05. The stop criterions are after 1000 generations or after 10000 solutions are found.

1) Results of the G-1 problems: Table 12 contains the solutions found by the three methods on the 8x8, 10x10 and 15x10 benchmarks. Note that the W_{td}, Max(W_k) and Makespan in the results tables represent total workload of all machines, the critical machine workload and the maximal completion time, respectively. From the results, the proposed method performs significantly better than the hybrid PSO & SA in solving all benchmarks.

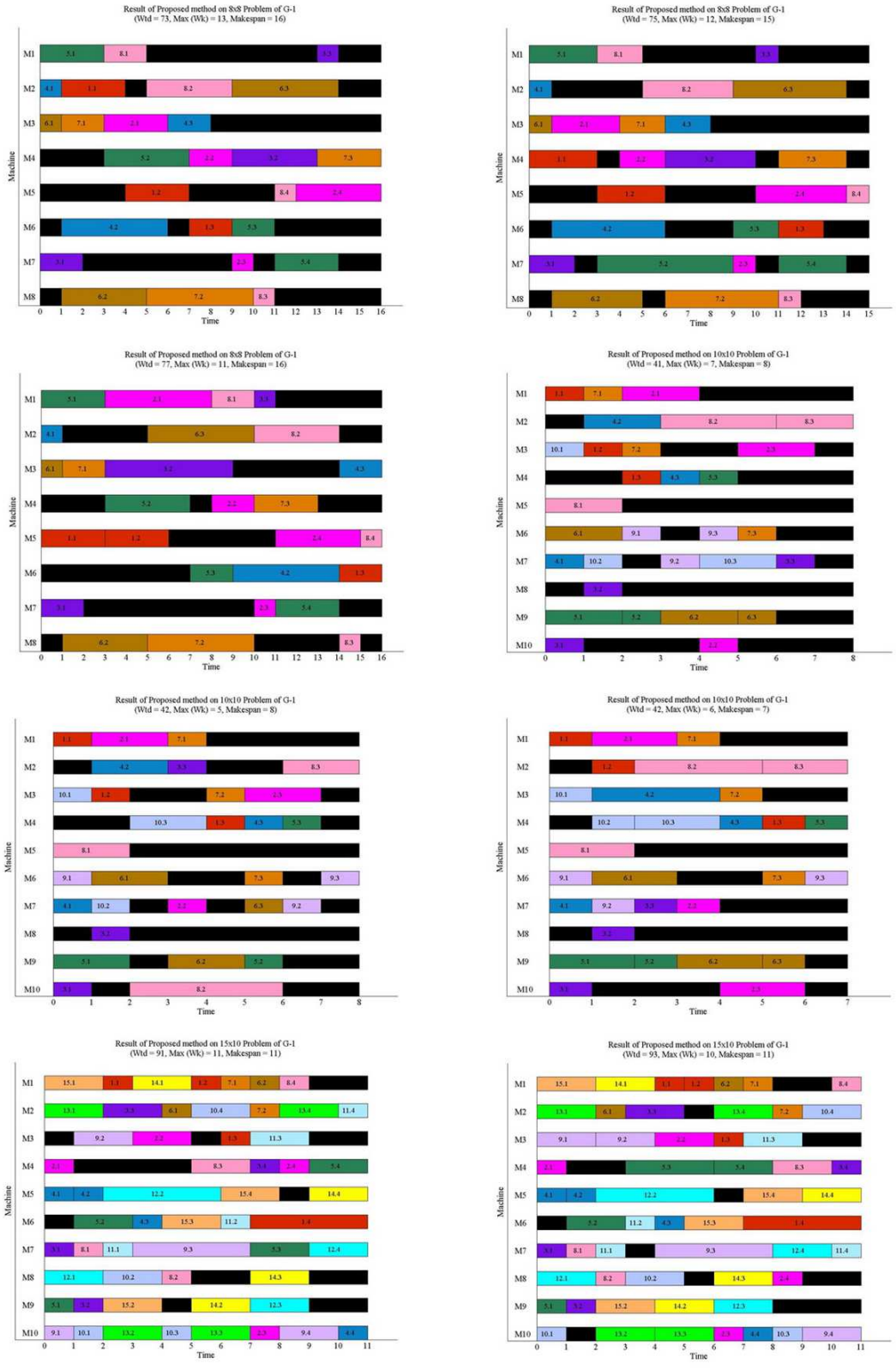

Fig. 16: Part of solutions found by proposed method for G-1 problems

Fig. 17: Part of solutions found by proposed method for G-2 problems

Table 12: Experiments results for G-1 benchmarks

Proposed Method										
Benchmark Results	8 x 8				10 x 10				15 x 10	
W_{td}	73	75	77	77	41	42	42	43	91	93
$Max(W_k)$	13	12	11	12	7	5	6	5	11	10
Makespan	16	15	16	14	8	8	7	7	11	11
Solution count	119	24	5	4	$> 10^4$	$> 10^4$	$> 10^4$	2680	$> 10^4$	$> 10^4$
MOEA-GLS [8]										
Benchmark Results	8 x 8				10 x 10				15 x 10	
W_{td}	73	75	77	77	41	42	42	43	91	93
$Max(W_k)$	13	12	11	12	7	5	6	5	11	10
Makespan	16	15	16	14	8	8	7	7	11	11
Solution count	1	1	1	1	1	1	1	1	1	1
Hybrid PSO & SA [5]										
Benchmark Results	8 x 8				10 x 10				15 x 10	
W_{td}	75		77		44				91	
$Max(W_k)$	12		13		6				11	
Makespan	15		16		7				12	
Solution count	1		1		1				1	
VNGA [9]										
Benchmark Results	8 x 8				10 x 10				15 x 10	
W_{td}	73	75	77	77	41	42	42	43	91	93
$Max(W_k)$	13	12	11	12	7	5	6	5	11	10
Makespan	16	15	16	14	8	8	7	7	11	11
Solution count	1	1	1	1	1	1	1	1	1	1

Table 13: Results for G-2 Benchmarks

Proposed Method								
Benchmark Results	4 x 5			10 x 7			15 x 10	
W_{td}	31	32	33	60	61	62	91	93
$Max(W_k)$	9	83	7	12	11	10	11	10
Makespan	16	16	16	16	15	15	23	23
Solution count	13	10	4	$> 10^4$	$> 10^4$	1097	$> 10^4$	$> 10^4$
MOEA-GLS [8]								
Benchmark Results	4 x 5			10 x 7			15 x 10	
W_{td}	32		33	60	61	62	91	93
$Max(W_k)$	8		7	12	11	10	11	10
Makespan	16		16	16	15	15	23	23
Solution count	1		1	1	1	1	1	1

Although it appears as if the proposed method performs similarly to the MOEA-GLS and VNGA on all benchmarks, the proposed method can find more solutions. In Ho's or Zhang's method, each optimal solution is mapped only to one schematic. The solutions of the G-1 problems found by the proposed method are shown in Fig. 16. Multiple solutions were discovered by the proposed method, but only some of them are exhibited.

2) Results of the G-2 problems: It was obvious from the results of G-1 problems (without release time constraints), that hybrid PSO & SA cannot find solutions located on/nearest to Pareto front. Thus, in this set of experiments, only the proposed method and MOEA-GLS were taken into comparison. Table 13 presents the solutions found by the two methods on the 4x5, 10x7 and 15x10 benchmarks. Note that the W_{td}, $Max(W_k)$ and Makespan in results tables also represent total workload of all machines, the critical machine workload and the time with all jobs finished, respectively. From the results, the proposed method produces better solutions than MOEA-GLS on the 4x5 benchmark. For the other two benchmarks, the proposed method performs similarly to the MOEA-GLS. However, the number of solutions found by the proposed method greatly surpasses MOEA-GLS. The solutions of the G-2 problems found by the proposed method are shown in Fig. 17. The proposed method found a mass of solutions, but only a few are exhibited.

The proposed method has the ability to find more/better solutions which are nearest to/located on the Pareto set in each benchmark.

7 Conclusions

In this paper, a solution characteristic extraction strategy is proposed to enhance genetic algorithms for multi-objective problems. It can significantly improve the searching abilities of chromosomes and find better solutions located on/nearest to the Pareto set. Two groups of benchmarks were adopted for testing through reasonable experimental conditions and the results are very reliable. The results of the experiments prove that the proposed method can provide users with more solution choices to meet their requirement.

Acknowledgement

This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 104-2221-E-161-003-MY2 and MOST 104-2221-E-131-033.

Appendance

Due to the page limitation, only part of the experiment results is listed in this paper. The complete experiment results can be found in http://web.csie.ndhu.edu.tw/sjyen/GAFJSP.htm

References

[1] A. A Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L Wang, "Heterogeneous computing: challenges and opportunities," *IEEE Computer*, vol. 26 (1993) 18-27.

[2] B. Hamidzadeh, L. Y. Kit, and D. J. Lilja, "Dynamic task scheduling using online optimization" *IEEE Trans. on Parallel and Distributed Systems*, vol. 11 (2000) 1151 – 1163.

[3] S. Baruah, "Partitioning real-time tasks among heterogeneous multiprocessors," in *Proc. of the 33rd International Conference on Parallel Processing*, (2004) 467-474.

[4] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *Proc. of the 10th International IEEE Real-Time and Embedded Technology and Applications Symposium*, (2004) 536-543.

[5] W. Xia, and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Computers and Industrial Engineering*, vol. 48, no. 2, (2005) 409-425.

[6] Z. Jia, H. Chen, and J. Tang, "An Improved Particle Swarm Optimization for Multi-objective Flexible Job-shop Scheduling Problem," in *Proc. of 2007 IEEE International Conference on Grey Systems and Intelligent Services*, (2007) 1587-1592.

[7] D. Luo, S. Wu, M. Li, and Z. Yang, "Ant Colony Optimization with Local Search Applied to the Flexible Job Shop Scheduling Problems," in *Proc. of IEEE Conference on ICCCAS 2008*, (2008) 1015-1020.

[8] N. B. Ho, and J. C. Tay, "Solving Multiple-Objective Flexible Job Shop Problems by Evolution and Local Search," *IEEE Trans. on System, Man, and Cybernetics-Part C: Applications and reviews*, vol. 38, no. 5, (2008) 674-685.

[9] G. Zhang, L. Gao, and Y. Shi, "A Novel Variable Neighborhood Genetic Algorithm for Multi-objective Flexible Job-Shop Scheduling Problems," in *Advanced Materials Research*, vol. 118-120, (2010) 369-373.

[10] Flexible Job Shop Problem, http://www.idsia.ch/~monaldo/fjsp.html.

[11] A. Coello Coello, "A comprehensive survey of evolutionary-based multiobjective optimization technique," Knowledge and Information Systems, vol. 1, no. 3, (1999) 269-308.

[12] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[13] L. B. Booker, D. E. Gokdberg, and J. H. Holland, *Classifier Systems and Genetic Algorithms*, Technical Report, no. 8, University of Michigan, 1978.

[14] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[15] Z. Michalewicz, T. Logan, and S. Swaminathan, "Evolutionary operations for continuous convex parameter spaces," in *Proc.* of the 3rd Annual Conference on Evolutionary Programming, (1994) 84-97.

Sheng-Ta Hsieh received the B.Sc. degree in electrical engineering from the National Taiwan University of Science and Technology in 2002 and received Ph.D. degrees in electrical engineering from National Dong Hwa University, Hualien, Taiwan, in 2007. He is currently an associate professor and chair in the department of Department of Communication Engineering, Oriental Institute of Technology, New Taipei City, Taiwan, R.O.C. His interest in research includes computational intelligence, internet of things(IoT), embedded systems, and signal processing.

Shi-Jim Yen received his B.S. in Computer Science and Information Engineering from Tamkang University, M.S. in Electrical Engineering from National Central University, and Ph.D. degree in Computer Science and Information Engineering from National Taiwan University, in 1991, 1993 and 1999, respectively. He is currently a Professor in the Department of

Computer Science and Information Engineering at the National Dong Hwa University, Hualien, Taiwan. He has specialized in artificial intelligence and computer games. In these areas, he has published over 100 papers in international journals or conference proceedings. He is a 6-dan Go player. He is an IEEE Senior Member. He served as a Program Chair of the 2015 IEEE Conference on Computational Intelligence and Games, and a Tournament Chair of 2010-2016 Conference on Technologies and Applications of Artificial Intelligence. He served as a Workshop Cochair of 2011 IEEE International Conference on Fuzzy Systems. He is the Chair of the IEEE Computational Intelligence Society (CIS) Emergent Technologies Technical Committee (ETTC) Task Force on Emerging Technologies for Computer Go since 2009. His team develops many strong board game programs including Go, Chinese chess, Dark chess, Connest6, and many puzzle games. These programs won the gold and other medals many times in Computer Olympiad, TAAI tournaments and TCGA tournaments since 2001. He got the Excellent Junior Researcher Project Award from Taiwan National Science Council in 2012 and 2013.

Chun-Ling Lin is currently an assistant professor with the Department of Electrical Engineering, Ming Chi University of Technology, New Taipei City, Taiwan. Her received a Ph.D. degree in electrical and control engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2012. Her interest in research includes EEG/ECG signal processing, expert system, and computational intelligence.

Shih-Yuan Chiu received his M.S. degree in electrical engineering from the National Dong Hwa University in 2007, and received his Ph.D. degrees in Computer Science and Information Engineering from National Dong Hwa University, Hualien, Taiwan, in 2013. He is currently an engineer of systems development center, Chung-Shan Institute of Science and Technology , Taoyuan City, Taiwan, R.O.C. His research interests include computer games, puzzle game solver and computational intelligence. He is also a 4-dan Go player.

Tsan-Cheng Su received his M.S. degree in Computer Science and Information Engineering from the National Dong Hwa University (NDHU), Taiwan in 2009. Currently, he is working as Ph.D. Candidate and Lecturer at the Department of Computer Science and Information Engineering, NDHU.