A Comprehensive Study of Genetic Algorithm for the Flowshop Scheduling Problem

Neelam Tyagi[1*], R.G.Varshney[2] and A.B.Chandramouli[3]

Abstract— In this paper, we introduced the methodology, operator's and concepts of a Genetic Algorithm. We described a Genetic Algorithm based heuristic for solving the flowshop scheduling problems. Flow-shop scheduling problem (FSSP) deals with the scheduling of a set of jobs that visit a set of machines in the same order. Heuristics play a major role for solving NP –hard combinatorial optimization problems. This paper describes a Genetic Algorithm based heuristic to makespan minimization on flowshop scheduling. We compared our heuristic with the NEH (Nawaz, Enscore, Ham) algorithm which is the most popular heuristic in the literature. The computational experience shows that the Genetic Algorithm approach provides competitive results for flowshop scheduling problems.

Index Terms—Flowshop scheduling, Crossover operators, Genetic Algorithm , makespan, Mutation operators, NEH Algorithm.

1 INTRODUCTION

THE flowshop scheduling has been a very active and prolific research area since the seminal paper of Johnson [49].

The flowshop scheduling problem is a production problem where a set of n jobs have to be processed with identical flow pattern on m machines. When the sequence of job processing on all machines is the same we have the permutation flowshop sequencing production environment. Since there is no job passing, the number of possible schedules for n jobs is $n!$.In scheduling problems we must determine the order or sequence for processing a set of jobs through several machines in an optimal manner. We study the flow shop problems considering the following assumptions:

(i) The operation processing times on the machines are known, fixed and some of them may be zero if some job is not processed on a machine.

(ii) Set-up times are included in the processing times and they are independent of the job position in the sequence of jobs.

(iii) At a time, every job is processed on only one machine, and every machine processes only one job.

(iv)The job operations on the machines may not be preempted.

Initial research concerning flowshop scheduling problem was done by Johnson [49]. Johnson described an exact algorithm to minimize makespan for the n-jobs two-machine flowshop scheduling problem. Later, algorithms, such as branch-and bound and beam search, that yield the exact solution for this problem were proposed. The flow shop scheduling problem that includes many jobs and machines is a combinatorial optimization problem for the NP-hard problem category. Therefore, near optimum solution techniques are preferred. Several heuristic approaches for the flow shop scheduling problem are developed. In recent years, metaheuristic approaches, such as simulated annealing, tabu search, and genetic algorithms, have become very desirable in solving combinatorial optimization problems because of their computational performance.

The rest of paper is organized as follows: Section 2 represents an introduction to GA. Section 3 and 4 describe the background, applications, methodology and different operators of Genetic Algorithms. In section 5 and 6 we explain the GA based and NEH heuristics respectively. Section 7 represent the model of GA. Computational result show in section 8 and finally section 9 represents the conclusion and suggestion for future direction.

1.1 Nomenclature

n	number of jobs
m	number of machines
p_{ij}	processing time of job i on machine j
p_c	probability of crossover
p_m	probability of mutation
C_{max}, C^*	makespan, optimal makespan value or lower bound value
$S(t)$	the population in the t-th generation

2 LITERATURE REVIEW

A significant research effort has been devoted for sequencing jobs in a flowshop with the objective of finding a sequence that minimizes the makespan. For problems with 2 machines, or 3

- [1]Graphic Era University, Dehradun, Uttarakhand, India *Corresponding Author: neelam24tyagi@gmail.com
- [2]Department of mathematics, Graphic Era University, Dehradun, Uttarakhand, India, E-mail: drrgvarshney@gmail.com
- [3]Department of mathematics, Meerut College, Meerut, Uttar Pradesh, India, E-mail: dr.abchandramouli@gmail.com

machines under specific constraints on job processing times, the efficient Johnson's algorithm obtains an optimal solution for the problem. The famous "Johnson's rule" is a fast **O(nlogn)** method for obtaining the optimal solution for the $F2/prmu/C_{max}$ (two machines) and for some special cases with three machines. Later, Palmer [11] presented a heuristic to solve the more general ***m*** -machine PFSP. Campbell, Dudek and Smith[21] develop another heuristic which is basically an extension of Johnson's algorithm to the ***m*** machine case. The list of heuristics is almost endless. Ruiz and Maroto[44] provided a comprehensive evaluation of heuristic methods and concluded that the famous NEH heuristic of Nawaz, Enscore and Ham[39] provides the best performance for the $F/prmu/C_{max}$ problem. As a matter of fact, the NEH is actively studied today as the recent papers of Framinan, Leisten and Rajendran [27], Dong, Huang and Chen [52], Rad, Ruiz and Boroojerdian [47] or Kalczynski and Kamburowski [42] show. However, since this scheduling problem is NP-hard (Garey, Johnson, & Sethi [40]) the search for an optimal solution is of more theoretical than practical importance. Therefore, since the 1960s a number of heuristic methods that provide near optimal or good solutions with limited computation effort have been proposed for flowshop sequencing. The performance of some earlier heuristics was evaluated by Dannenbring[9].

2.1 Heuristic methods can be classified according to two major categories

Constructive methods*:* The constructive algorithms obtain directly a solution for the scheduling problem, i.e. a ***n***-job sequence, by using some procedure which assigns to each job a priority or an *index* in order to construct the solution sequence (see for example, Campbell, Dudek, & Smith[21], Dannenbring[9], Davoud Pour[20], Gupta[28], Kalczynski & Kamburowski[43], Koulamas[3], Nagano & Moccellin[41], Nawaz, Enscore, & Ham[39], Palmer[11].

Improvement method*:* An improvement method starts from a given initial solution, and looks for a better one normally by using some neighbourhood search procedure.

- Metaheuristics can also be considered as improvement heuristics. Regarding metaheuristics, there is also a vast literature of different proposals for the PFSP under different criteria. Metaheuristics can also be considered as improvement heuristics. Within this type of techniques we find genetic algorithms (GAs), simulated annealing (SA), tabu search (TS) and other procedures or hybrid methods. The first proposed metaheuristics for the permutation flowshop scheduling problem (PFSP) are the simulated annealing algorithms by Osman and Potts [23] and Ogbu and Smith [16]. Grabowski and Wodecki [24] demonstrated different tabu search approaches. While Genetic Algorithms are presented in Reeves [6] and in Ruiz, Maroto and Alcaraz [46]. Other algorithms are the path-based method of Werner [17] or the iterated local search (ILS) of Stutzle [51].
- Recentaly Other metaheuristics like Ant Colony Optiomization, a very fast Tabu Search (TS) approach, Scatter Search, An updated and comprehensive review of flowshop heuristics and metaheuristics, Discrete Differential Evolution, Particle Swarm Optimization or Iterated Greedy are presented in Rajendran and Ziegler[8], Grabowski and Wodecki[24], Nowicki and Smutnicki[13], Ruiz and Maroto[44], Onwubolu and Davendra[18], Tasgetiren et al.[36] and Ruiz and Stützle[45], respectively.
- Recent and high performing approaches include parallel computing methodologies presented in Vallada and Ruiz[14]. Apart from makespan minimization, the PFSP has been studied under many other criteria. For example, Vallada, Ruiz and Minella [15] review 40 heuristics and metaheuristics for tardiness-related criteria. Similarly, a recent review of multiobjective approaches is given in Minella, Ruiz and Ciavotta [19].

3 GENETIC ALGORITHMS

Genetic algorithms were developed by Holland in [25]. The genetic algorithm (GA) is a search technique based on the mechanics of natural genetics and survival of the fittest (Goldberg [10]). GA simulates the biological processes that allow the consecutive generations in a population to adapt to their environment. The adaptation process is mainly applied through genetic inheritance from parents to children and through survival of the fittest. The genetic algorithm object determines which individuals should survive, which should reproduce, and which should die. Since genetic algorithms (GAs) are adaptive and flexible, the GAs were shown to be successfully applied to several optimization problems. For example, they have been applied to routing, scheduling, adaptive control, game playing, cognitive modeling, transportation problem, traveling salesman problems, optimal control problems, database query optimization, etc.

The Gas are stochastic search techniques whose search algorithms simulate natural phenomena (biological evolution). Genetics is a biological term. Biologically, genes of a good parent produce better offspring The basic idea of the GAs is that the strong tend to adapt and survive while the weak tend to die. One of the strengths of GAs is that they use past information to direct their search with the assumption of improved performance. The formal description of the GA which was provided by Grefenstette is as follows:

… A genetic algorithm is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each temporal increment (called a generation), the structures in the current population are rated for their effectiveness as domain solution, and on the basis of these evaluations, a new population of candidate solutions is formed using specific genetic operators such as reproduction, crossover, and mutation. (Grefenstette [26])-

To successfully apply a GA to solve a problem one needs to determine the following (Premalatha *et al*. [29]):

- The representation of possible solutions, or the chromosomal encoding.
- The fitness function which accurately represents the value of the solution.
- Genetic operators (selection, crossover and Mutation)

have to employ and the parameter values (population size, probability of applying operators, etc.), that are suitable.

3.1 GA are different from more normal optimization and search procedures in four ways

- ✓ GAs work with a coding of the parameter set (as mentioned above), not the parameters themselves.
- ✓ GAs search from a population of points, not a single point.
- ✓ GAs does not use derivatives or other auxiliary knowl-edge.
- ✓ GAs use probabilistic transition rules, not deterministic rules.

It also records statistics and decides how long the evolution should continue. Figure 1 illustrates the simple genetic algorithm (Premalatha *et al*. [29]).

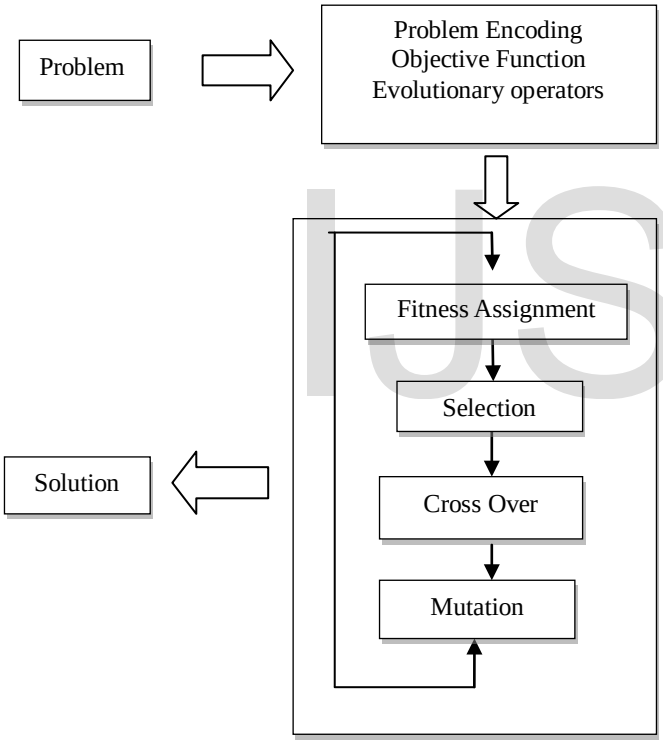

Fig. 1. Simple genetic algorithm (Premalatha et al., 2009)

3.2 Background of GA

Many human inventions were inspired by nature. Artificial neural networks is one example. Another example is Genetic Algorithms (GA). GAs search by simulating evolution, starting from an initial set of solutions or hypotheses, and generating successive "generations" of solutions. This particular branch of AI was inspired by the way living things evolved into more successful organisms in nature.

The main idea is survival of the fittest, a.k.a. natural selection. A chromosome is a long, complicated thread of DNA (deoxyribonucleic acid). Hereditary factors that determine particular traits of an individual are strung along the length of these chromosomes, like beads on a necklace. Each trait is coded by some combination of DNA (there are four bases, A (Adenine), C (Cytosine), T (Thymine) and G (Guanine). Like an alphabet in a language, meaningful combinations of the bases produce specific instructions to the cell. Changes occur during reproduction. The chromosomes from the parents exchange randomly by a process called crossover.

Therefore, the offspring exhibit some traits of the father and some traits of the mother. A rarer process called mutation also changes some traits. Sometimes an error may occur during copying of chromosomes (mitosis). The parent cell may have -A-C-G-C-T- but an accident may occur and changes the new cell to -A-C-T-C-T- . Much like a typist copying a book, sometimes a few mistakes are made. Usually this results in a nonsensical word and the cell does not survive. But over millions of years, sometimes the accidental mistake produces a more beautiful phrase for the book, thus producing a better species.

3.3 Methodology

The general procedures of the GA are as follows:

1. Initialize a population of binary or non-binary chromosomes.
2. Evaluate each chromosome in the population using the fitness function.
3. Select chromosome to mate (reproduction).
4. Apply genetic operators (crossover and mutation) on chromosome selected.
5. Put chromosomes produced in a temporary population.
6. If the temporary population is full, then go to step 7. Otherwise , go to step 3.
7. Replace the current population with the temporary population.
8. If termination criterion is satisfied, then quit with the best chromosome as the solution for the problem. Otherwise , go to step 2

3.4 Simple Genetic Algorithm

/*Algorithm GA */

Formulate initial population
Randomly initialize population
Repeat

Evaluate objective function
Find fitness function
Apply genetic operators

Reproduction
Crossover
Mutation

Until stopping criteria

Fig. 2. The Working Principle of a Simple Genetic Algorithm

3.5 Genetic Algorithms are composed of three operators

1. Reproduction,

2. Crossover,
3. Mutation

Reproduction is a process in which individual strings are copied according to their objective function values (biologists call this function the fitness function). We can think of the fitness function as some measure of profit, utility or goodness that we want to maximize. This operator, of course, is an artificial version of natural selection.

Crossover is the partial exchange of information using a cross-site chosen at random. First strings in the mating pool are mated at random then new strings are created by swapping the selected elements of the string (Figure 3).

Mutation is the occasional (with small probability) random alteration of the value of a string position. In the traditional representation of GA this means changing a 1-a 0 and vice versa. By itself, mutation is a random walk through the string space. The mutation operator plays a secondary role on the GA.

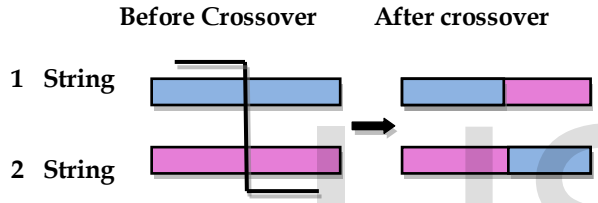

Fig. 3. Crossover

Application of Genetic Algorithms

The application of genetic algorithms to the flow shop scheduling problem has been widely studied.

- Chen et al. [4] developed one of the earliest genetic algorithms for the flow shop scheduling problem with the makespan minimization criterion.
- Reeves [6] also described the concept of genetic algorithms and applied it solving the flow shop scheduling problem with makespan as a criterion.
- Murata et al. [50] examined the performance of genetic algorithms in order to specify some genetic operators and parameters for the flow shop scheduling problem. They then proposed two hybrid genetic algorithms to improve the performance of the genetic algorithm. One is the genetic local search algorithm and the other is a genetic simulated annealing algorithm. They also introduced some modifications of search mechanisms in these hybrid genetic algorithms.
- Cotta and Troya [2] studied different representations for the flow shop scheduling problem using forma analysis. They proposed some new operators that run on these representations.
- Reeves and Yamada [5] re-considered the implementation of a genetic algorithm for the flow shop scheduling problem using the representative neighborhood and path re-linking.
- Wang et al. [31] presented a class of order-based genetic algorithms for the flow shop scheduling problem. This algorithm borrows from the idea of ordinal optimization to ensure the quality of the solution found with a reduced computation effort. It is applied to evolutionary search mechanisms and learning capabilities of genetic algorithms to effectively perform exploration and exploitation.
- Wang and Zheng [31] proposed an effective hybrid heuristic for the flow shop scheduling problem. They incorporated the NEH heuristic into the random initialization of a genetic algorithm, used multicrossover operators acting on the divided subpopulations, and replaced mutation by the simulated annealing metropolis sample process with multiple neighbor state generators.
- Iyer and Saxena [48] improved the standard implementation of the genetic algorithm by tailoring the various genetic algorithm operators to suit the structure of the problem.
- Wang et al. [34] first formulated the determination of optimal genetic control parameters. Then the ordinal optimization and the optimal computing budget allocation techniques are applied to determine the best genetic control parameters among all the alternative parameter combinations.
- Ruiz et al. [46] proposed a robust genetic algorithm and a rapid hybrid implementation for solving the permutation flow shop scheduling problem. These algorithms use new genetic operators, advanced techniques like hybridization with local search, an efficient population initialization, and a new generational scheme.
- Wang and Zhang [32] presented a novel and systematic approach based on ordinal optimization and optimal computing budget allocation techniques to determine the optimal combinations of genetic operators for flow shop scheduling problems.
- Zhang et al. [35] proposed an adaptive genetic algorithm with multiple operators for the flow shop scheduling problem. This adaptive genetic algorithm uses multiple crossover and mutation operators in an adaptively hybrid sense, according to their contribution to the search process.

4 Genetic Algorithm for Flowshop Scheduling Problems

Flowshop scheduling is one of the most well-known problems in the area of scheduling. Various approaches to this problem have been proposed since the pioneering work of Johnson [49]. GAs has been applied to combinatorial optimization problems such as the traveling salesman problem and scheduling problems (see for example, Fox & McMahon [1], Ishibuchi, Yamamoto, Murata, & Tanaka [22]. When applying GAs to a scheduling problem there is an obvious practical difficulty. We need a different string representation and genetic operators. These are shown below in detail

- ***Representation of structure*** **:** The traditional representation of GA which contains 0's and l's does not work for scheduling problems. In order to apply any GA to a scheduling problem, a structure can be described as a sequence of the jobs in the problem.

Elements of Genetic Algorithms

4.1 Population initialization and population size

The first element of the GAs is the size of population and how to generate the initial population. The initial population of chromosomes can be generated randomly or by using some heuristics that are suitable for the problem considered. The determination of the population size is a crucial element in the GAs. Selecting a very small population size increases the risk of prematurely converging to a local optimal. Large popula-

tion sizes increases the probability of converging to a global optimal, but it will take more time to converge. In most of the GA applications, the population size was maintained at a constant. Most GAs in other contexts assumes that the initial population is chosen completely at random. But it is a good idea to construct the initial population drawing on other heuristics in the literature. This enables us to arrive at the final solution more quickly. Reeves' used the NEH Algorithm to generate the initial population. He obtained one solution from this heuristic and generated others at random. Chen et al, used the CDS algorithm (a heuristic developed by Campbell, Dudek and Smith) to construct the initial population. Chen et al also stated that the initial population for their GA would be generated using other well known heuristics (for example, Dannenbring's method or a job insertion-based method). We use the ***m***-1 schedules produced by the CDS method and one schedule produced by using the Dannenbring method to generate an initial population. This operator selects a member at random and swaps two randomly selected positions of the member to generate a new member for the initial population. This procedure will be repeated until the number of the members is equal to population size. Chen et al, generated some trial examples and run their heuristics with different population sizes for each example; They found that the population with size more than 60 cannot guarantee better results than the population with size equal to 60. Therefore, we decided to use 60 as a population size of our heuristic.

4.2 Fitness function

The second element of the GAs is the fitness function, which is very important of the GAs process of evolution. The GA without fitness function is blind because the GA directs its search using historical data which are the fitness values of the chromosomes. The GA will use the fitness values of each chromosome to determine if the chromosome can survive and produce offspring, or die.

There are different criteria used as fitness values of a structure. The most popular of these are makespan (maximum completion time) and total flow time. We use the makespan criterion in our heuristic. For a maximization problem, the measure of performance generally constitutes the fitness function. However, our objective is to minimize the makespan. For minimization problems, the method to determine the fitness function differs from the maximization problems. Fitness function of the strings can be calculated as follows:

$$f(S_i(t)) = \max\{C(S_i(t)\} - C(S_i t) \quad (1.1)$$

where $S_i(t)$ is the *i*th string in *t*th generation, $C(S_i(t))$ is the makespan of $S_i(t)$ and $f(S_i(t))$ is the fitness function of $S_i(t)$. Therefore, the probability of selection for a schedule $P(S_i(t))$ with lower makespan is high (Equation 1.2).

$$P(S_i(t)) = f(S_i(t)) \ / \ \Sigma f \quad (1.2)$$

This is also the criterion used for the selection of parents for the reproduction of children.

Genetic Algorithm Operators

4.3 Reproduction (or Selection) of chromosomes

The selection of chromosomes to reproduce is the third element of the GA. Reproduction (or selection) is an operator that makes more copies of better strings in a new population. Reproduction is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. This is one of the reasons for the reproduction operation to be sometimes known as the selection operator. Thus, in reproduction operation the process of natural selection causes those individuals that encode successful structures to produce copies more frequently. To sustain the generation of a new population, the reproduction of the individuals in the current population is necessary. For better individuals, these should be from the fittest individuals of the previous population. This is very important element in the GA because it plays an important role in the convergence of the GA. If the selection process is always biased to only accept the best chromosome, the algorithm will quickly have a population of almost the same chromosome which will cause the GA to converge to a local optimum. Several selection methods have been employed by several researchers to select among the best performers. Some of these methods are: the proportional selection scheme; the roulette wheel selection; deterministic selection; ranking selection; tournament selection, etc.

- ***Roulette wheel selection***

Roulette wheel selection is chosen, where the average fitness of each chromosome is calculated depending on the total fitness of the whole population. The chromosomes are randomly selected proportional to their average fitness.

Roulette wheel selection is summarized in the following steps,

- *Step1*. Let the *pop-size*, number of strings in *pop.*
- *Step2*. *nsum*, sum of all of the fitness values of the strings in *pop*; form *nsum* slots and assign string to the slots according to the fitness value of the string.
- *Step3*. Do step 4 (*pop-size* -1) times.
- *Step4*. Generate a random number between 1 and *nsum*, and use it to index into the slots to find
- *Step5*. Add the string with the highest fitness value in *pop* to *newpop*. the corresponding string; add this string to *newpop*

- ***Tournament selection*** [10]**.**

In Tournament Selection, predetermined numbers of chromosomes are randomly selected from the population and the chromosome with the best fitness value is considered to be regenerated. Here selection is based on a competition within a subset of the population.

4.4 Crossover

Crossover is used as the main genetic operator and the performance of a GA is heavily dependent on it. It's a fourth element of GA. A crossover operator is used to recombine two strings to get a better string.It is important to note that no new strings are formed in the reproduction phase. In the crossover operator, new strings are created by exchanging information among strings of the mating pool. A crossover operator is mainly responsible for the search of new strings even though mutation operator is also used for this purpose sparingly.

Assume that in the initial population there are two parents which are:

Parent 1: 6-7-5-4-2-3-1-8 and
Parent 2: 8-4-3-1-7-5-2-6

A single- position crossover method is performed on the two parents, where the single- position crossover is denoted by "|" as shown below.

Parent 1: 6-7-5-4-2-|3-1-8 and
Parent 2: 8-4-3-1-7-|5-2-6

Child 1: 6-7-5-4-2-5-2-6 and
Child 2: 8-4-3-1-7-3-1-8

It is obvious that both of children represent infeasible sequences because both of them have five jobs out of eight jobs, and each has three duplicated jobs. Therefore to solve this infeasibility problem, several crossover methods that produced feasible chromosomes were proposed by several researchers:

1. Order crossover (OX) by Davis [30].
2. Partially Mapped Crossover (PMX) by Goldberg and Lingle.
3. Sub-sequence-Swap crossover (SSX) and Sub-sequence-Chunk crossover (SCX) by Grefenstette et al. [26].
4. Cycle Crossover (CX) by Oliver, Smith, and Holland.
5. Edge Recombination Crossover (ERX) by Whitley, Starkweather, and Shaner [12].
6. Linear Order Crossover (LOX) by Falkenauer and Bouffouix.
7. Order-based Crossover (OBX) and Position-based Crossover (PBX) by Syswerda.
8. Enhanced edge recombination Crossover (EERX) by Starkweather et al.
9. Similar Job Order Crossover (SJOX) by Ruben Ruiz et al. [44].

Some of these crossover operators are briefly explained in the following:

4.4.1. Linear order crossover (LOX)

LOX, initially suggested by Falkenauer and Bouffouix, works as follows:
Step 1: Select a subsequence of operations from one parent at random.
Step 2: Produce a proto-offspring by copying the subsection sequence into the corresponding positions of it.
Step 3: Delete the operations which are already in the subsequence from the second parent.The resulted sequence of operations contains the operations that the proto-offspring needs
Step 4: Place the operations into the unfixed positions of the proto-offspring from left to right according to the order of the sequence to produce an offspring. This procedure is illustrated in figure 4.

Crossover LOX tries to preserve as much as possible both the relative positions between genes and the absolute positions relative to the extremities of parents

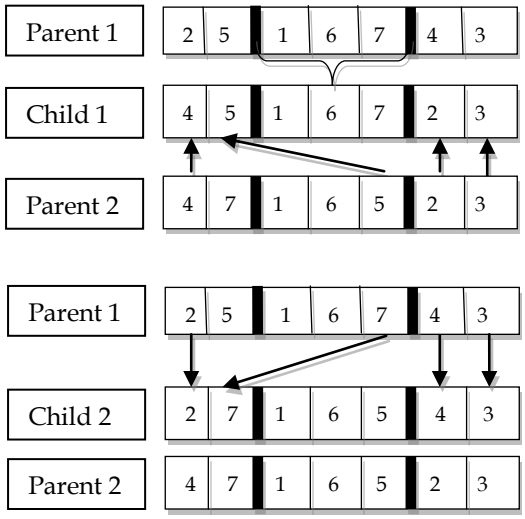

Fig. 4. LOX Crossover operator.

4.4.2. Position Based Crossover (PBX)

Step 1: Select a set of positions from one string at random,
Step 2: Produce a new string by copying the symbols on these positions into the corresponding positions in the new string,
Step 3: Delete the symbols already selected from the second string. The resulting sequence contains only the symbols that the new string needs,
Step 4: Place the symbols into unfixed positions in the new string from left to right according to the order of the sequence used to produce one offspring. First, it is generated a random mask and then exchanged relative genes between parents according to the mask.

4.4.3 Order crossover (OX).

The offspring inherits the elements between the two crossover points from the selected parent in the same order and position as they appear in the parent. The remaining elements are inherited from the alternate parent in the order in which they appear in that parent, beginning with the first position following the second crossover point and skipping over all elements already present in the off spring.

4.4.4. Partially Mapped Crossover (PMX)

A parent and two crossover sites are selected randomly and the elements between two string positions in one of the parents are directly inherited by the offspring. Each element between the two crossovers points in the alternate parent are mapped to the position held by this element in the first parent. Then the remaining elements are inherited from the alternate parent.
Step 1: Two points are randomly selected for dividing the parents. The section of the parents between these two points is called the mapping section (figure 5).
Step 2: Exchanges the mapping section of the parent to the

offspring. i.e. mapping section of the first parent is copied into the second offspring and so on.

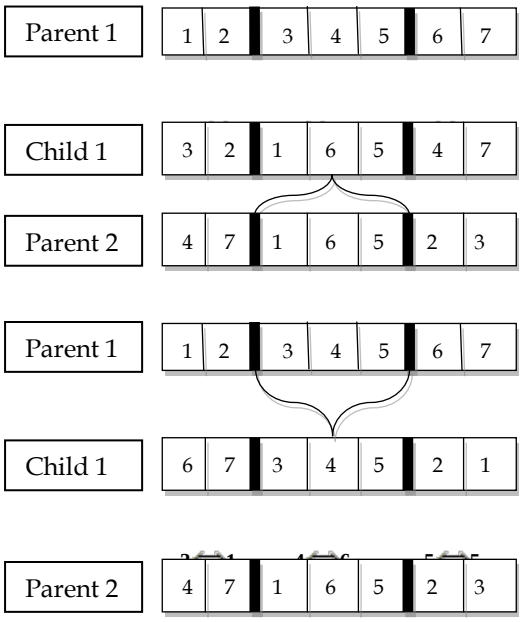

Fig. 5. Partially mapped crossover

Step 3: Define one-to-one mapping between genes of mapping section of the two parents. For the above example mapping is as follows (figure 6).

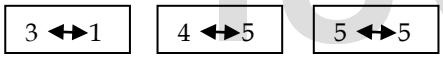

Fig. 6. One-to-one mapping.

4.4.5 Cycle Crossover (CX)

The cycle between strings is fined, the symbols in the cycle are coped to a new string, the remaining symbols are determined for the new string by deleting the symbols and the remaining symbols are fulfilled with the new string.

4.4.6 Order-based Crossover (OBX)

A set of positions is selected randomly; the order of symbols in the selected positions is imposed on the corresponding symbols in the other string. Mutation operator plays a very important role in GAs and it helps maintain diversity in the population to prevent premature convergence.

Six mutation operators are examined in the GA to minimize the makespan in HFS. These are neighborhood based, adjacent two job change, arbitrary two job change, arbitrary three job change, shift change and inversion mutation operator.

4.4.7 Similar job order crossover (SJOX)

SJOX crossover is based on the idea of identifying and maintaining building blocks in the offspring (Ruben Ruiz et al. [44]). In this way similar blocks or occurrences of jobs in both parents are passed over to child unaltered. If there are no similar blocks in the parents the crossover operator will behave like the single-point order crossover.

4.5 Mutation

Mutation is nearly always regarded as an integral part of a GA. Mutation generates an offspring solution by randomly modifying the parent's feature. It helps to preserve a reasonable level of population diversity, and provides a mechanism to escape from local optima. For each child obtained from crossover, the mutation operator is applied independently with a probability p_m.

Mutation adds new information in a random way to the genetic search process and ultimately helps to avoid getting trapped at local optima. It is an operator that introduces diversity in the population whenever the population tends to become homogeneous due to repeated use of reproduction and crossover operators. Mutation may cause the chromosomes of individuals to be different from those of their parent individuals. Mutation in a way is the process of randomly disturbing genetic information. They operate at the bit level; when the bits are being copied from the current string to the new string, there is probability that each bit may become mutated. This probability is usually a quite small value, called as mutation probability p_m.

A coin toss mechanism is employed; if random number between zero and one is less than the mutation probability, then the bit is inverted, so that zero becomes one and one becomes zero. This helps in introducing a bit of diversity to the population by scattering the occasional points. This random scattering would result in a better optima, or even modify a part of genetic code that will be beneficial in later operations. On the other hand, it might produce a weak individual that will never be selected for further operations.

The need for mutation is to create a point in the neighborhood of the current point, thereby achieving a local search around the current solution.

The mutation is also used to maintain diversity in the population. For example, the following population having four eight bit strings may be considered:

0 1 1 0 1 0 1 1
0 0 1 1 1 1 0 1
0 0 0 1 0 1 1 0
0 1 1 1 1 1 0 0

It can be noticed that all four strings have a 0 in the left most bit position. If the true optimum solution requires 1 in that position, then neither reproduction nor crossover operator described above will be able to create 1 in that position. The inclusion of mutation introduces probability p_m of turning 0 into 1.

In traditional GAs, mutation is applied by flipping each element of the structure from 1 to 0 (or vice versa) with a small probability. In sequence representation mutation needs to be defined differently. Mutation operator plays a very important role in GAs and it helps maintain diversity in the population to prevent premature convergence. Some mutation operators are examined in the GA to minimize the

makespan in Flowshop Scheduling. These are neighborhood based, adjacent two job change, arbitrary two job change, arbitrary three job change, shift change and inversion mutation operator etc.

4.5.1 Exchange mutation

Exchange mutation was a simple exchange of two elements of the structure, chosen at random.

- Before mutation : 4 **6** 2 8 5 1 **3** 7
- After mutation : 4 **3** 2 8 5 1 **6** 7

4.5.2 Inversion Mutation

It can be seen from Fig. 7 that the inversion mutation selects two positions at random and then swaps the genes on these positions.

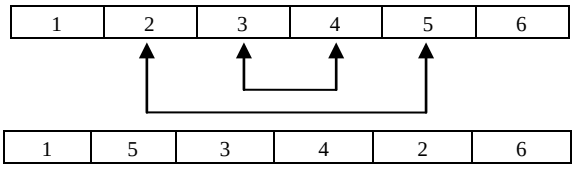

Fig. 7. The Inversion mutation operators

4.5.3 Shift mutation

Sift mutation was a shift of one element (chosen randomly) a random number of places to the right or left.

Before mutation: 4 6 2 8 5 1 3 7
After mutation : 4 3 6 2 8 5 1 7

Some experiments shows that shift mutation seemed to be better than exchange mutation (see Holland').

4.6 Generations (iteration)

Now that there is no practicable rule to set suitable stopping condition and it is also impossible for GA to evolve with too long time in real application, the usual way is to set a limit to a number of generations. These three operators are simple and straightforward. The reproduction operator selects good strings and the crossover operator recombines good sub-strings from good strings together, hopefully, to create a better sub-string. The mutation operator alters a string locally expecting a better string. Even though none of these claims are guaranteed and/or tested while creating a string, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be increasingly emphasized. Further insight into these operators, different ways of implementations and some mathematical foundations of genetic algorithms can be obtained from GA literature. Application of these operators on the current population creates a new population. This new population is used to generate subsequent populations and so on, yielding solutions that are closer to the optimum solution. The values of the objective function of the individuals of the new population are again determined by decoding the strings. These values express the fitness of the solutions of the new generations. This completes one cycle of genetic algorithm called a generation. In each generation if the solution is improved, it is stored as the best solution. This is repeated till convergence.

We use the number of generations (iteration) as the termination criterion. If the number of generations is low the probability of finding the best result is low. Otherwise if the number of generations is too high, the iteration time is too long. Figure 8 shows the experiments for improvement of the makespan according to the generations. We conducted some experiments by solving different sizes of problem at 35 generations and we found out that the makespan become stable after 20 generations. An empirical number of generations is 35. If the size of problems is larger than 30 x 30 and processing time interval is larger than 1-20, the number of generations may be increased. Therefore, we chose 20 as the termination criterion in our heuristic.

5 Genetic Algorithm based heuristic

Now, we are describes GA-based heuristic for the flowshop problems.

Step 1: Determine the initial population S(0) as described in an earlier section. The size of the population is 60. $t = 0$, NG=0

Step 2: Calculate the fitness value, $f(S_i(t))$, of each string for population. (See Equation (1.1))

Step 3: Calculate the selection probability, $P(S_i(t))$,of each string for population.(See Equation 1.2)

Step 4: Select a pair of strings (parents) according to selection probabilities of the members of S(t) (using random numbers).

Step 5: Constitute the new strings (children) by applying the LOX operator to the parents.

Step 6: Apply the shift mutation to the children with probability 0.05 ($P_m = 0.05$).

Step 7: Put the new strings in S(t+l). If the size of population S(t + 1) = 60 then NG = NG + 1 and go to Step 8, else go to Step 4.

Step 8: If NG = 20 then stop, else go to Step 2.

6 NEH Heuristic

The NEH heuristic was proposed by Nawaz et al [39] to solve the *m*-machine flow shop problem of minimizing makespan. The heuristic is based on the assumption that a job with more processing time on all machines will be given higher priority while a job with less processing time on all machines will receive lower priority. Accordingly, the two jobs with highest processing times are determined from the *n*-jobs problem. The best partial sequence for these two jobs is found by considering the two possible partial schedules. The relative positions of these two jobs with respect to each other are fixed in the remaining steps of the heuristic. Next, the job with the third highest processing time is determined and three partial sequences are tested in which this job is placed at the beginning, middle, and end of the partial sequence found before. The best partial sequence fixes the relative positions of these three jobs in the remaining steps of the heuristic. This procedure is repeated until all jobs are fixed and scheduled.

7 A Model Representation of Genetic Algorithms

A set of genetic operators such as reproduction (selection) and recombination (crossover and mutation) is applied to create new and better solutions (off springs) from the individuals of the current population and the solutions are steadily improved from generation to generation. The structure of GAs is given in Fig. 8.

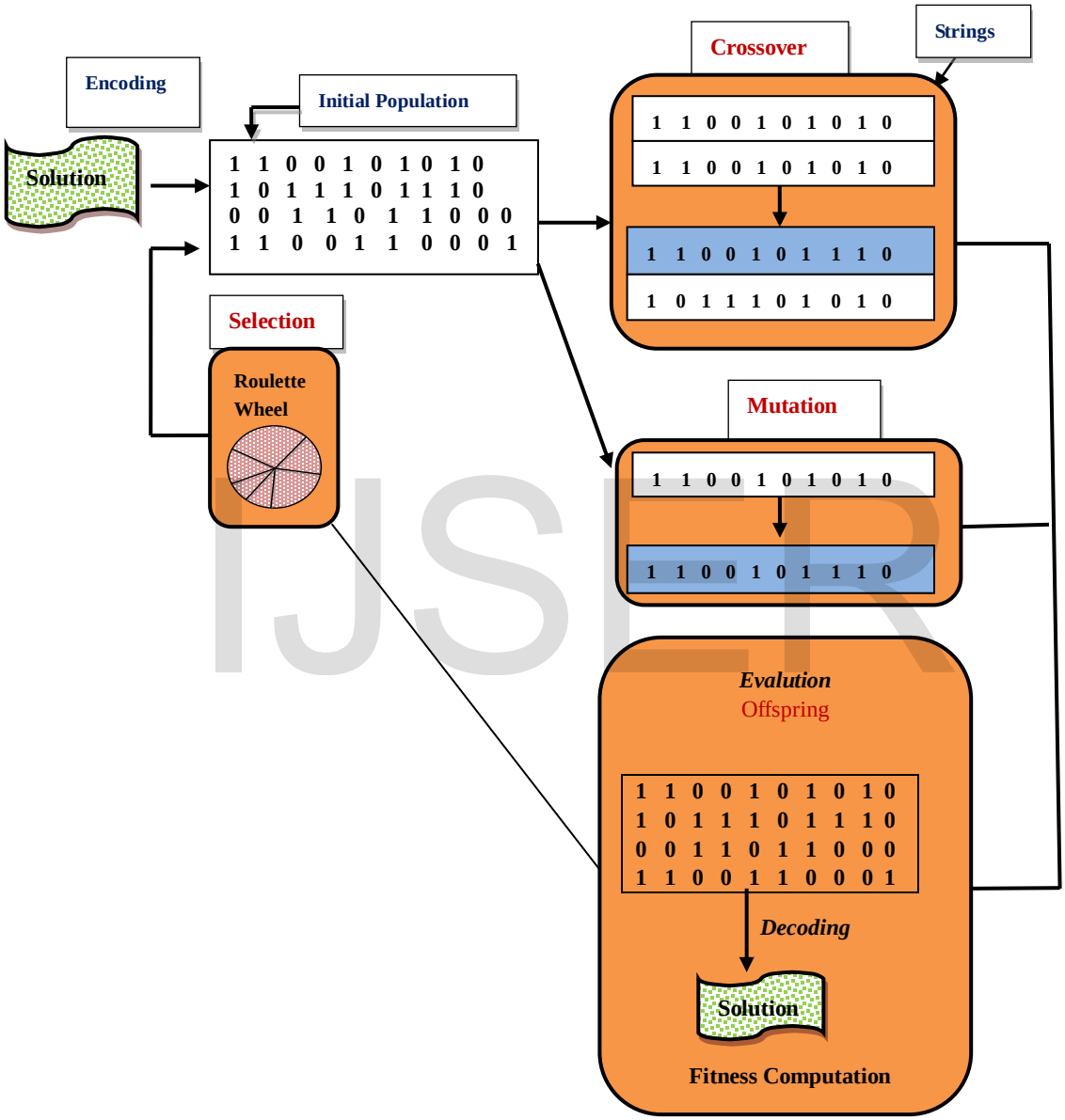

Fig. 8. The fundamental cycle and operations of basic GAs (Gen and Cheng [37])

8 Computational Results

In order to examine the effectiveness of the GA-based heuristic, one comparison was made over a wide range of jobs and machines. We compared our heuristic with the NEH (Nawaz, Enscore, Ham) Algorithm which is the most popular heuristic in the literature. NEH is a 20-year-old heuristic but most researchers still compare their heuristic with NEH or they include NEH in their algorithms. Armentano et al, showed the improvement percentage of tabu search with diversification and intensification compared to NEH algorithm. Koulamas [3] proposed a new heuristic called HFC (Heuristic Flowshop scheduling with C_{max} objective) and compared HFC with NEH algorithm. Ronconi26 also compared his MM (MinMax) algorithm with NEH. Framinan et al, showed the excellent performance of the NEH in their algorithm.

They proposed a heuristic for mean/total flowtime minimization in permutation flow shops. The heuristic exploits the idea of 'optimizing' partial schedules, already present in the NEH heuristic with respect to makespan minimization.

TABLE 1: RELATIVE PERFORMANCE OF GA (C_{GA}/C_{NEH})

The processing times were randomly sampled from a uniform distribution ranging from 1-20. Using this range enables us to compare our genetic algorithm with the Chen et a119h heuristic that uses the same range of processing times. Both heuristics were programmed in PASCAL and run on a Pentium IV (256 MB RAM) computer. In all, 230 problems were generated for 23 different combinations of job size and number of machines. It was not possible to solve problems larger than 40 x 40 because of software and machine limitations. This is caused by the build up computer memory requirements from the sizable GA population and the operations being carried out on it. The result of the two comparisons are presented in the following tables. In Table 1 the relative performance of the GA-based heuristic to the NEH was computed by CGA/CNEH where the C refers to the average makespan of the problems in the combination. Problems, used for calculation are the same as in Table 2 (200 runs were performed).

n x m	Generated problems	Advantage GA	Even	Advantage NEH	% GA	% NEH
8*5	10	8	2	0	100	20
8*10	10	4	6	0	100	60
8*15	10	8	2	0	100	20
8*20	10	8	1	1	90	20
10*5	10	4	4	2	80	60
10*10	10	7	2	1	90	30
10*15	10	6	2	2	80	40
10*20	10	7	3	0	100	30
15*5	10	9	0	1	90	10
15*10	10	7	3	0	100	30
15*15	10	5	1	4	60	50
15*20	10	7	0	3	70	30
20*5	10	3	2	5	50	70
20*10	10	6	1	3	70	40
20*15	10	6	1	3	70	40
20*20	10	5	0	5	50	50
30*5	10	1	3	6	40	90
30*10	10	5	0	5	50	50
30*15	10	4	0	6	40	60
30*20	10	2	2	6	40	80
30*30	10	8	0	2	80	20
35*35	10	9	0	1	90	10
40*40	10	10	0	0	100	0
Total	230	139	35	56	76	40

TABLE 2: COMPARISON OF GA WITH THE NEH

In Table 2, the first column is the pairing of the number of jobs, ***n***, and the number of machines, ***m***. The second column is the number of generated problems for the pairing. The third column and fifth column illustrate the number of times the best solution was obtained by the heuristic used, respectively. The fourth column shows the number of times that two heuristics in a comparison give the same makespan. The last two columns show the percentage of success of each heuristic, the

	m			
n	5	10	15	20
8	0.981	0.988	0.984	0.989
10	0.991	0.983	0.991	0.989
15	0.988	0.972	0.996	0.987
20	1.001	0.995	0.994	0.999
30	1.007	0.998	1.001	1.005

total number of times that the heuristic gives the best solution (number of advantage + number of even) divided by the number of generated problems. (The method of Widmer and Hertz.) According to the results in Table 1 the GA-based heuristic obviously yields better average makespan than the NEH.

Table 2 shows that in the 230 generated examples, the NEH gets better results than the GA-based heuristic only 56 times out of 230, while the GA-based heuristic is better 139 times out of 230. The NEH and the GA-based heuristic give the same results 35 times out of 230.

9 CONCLUSION AND DIRECTIONS FOR FUTURRE SEARCH

In this paper, we introduced the fundamental modal and described a GA-based heuristic for solving the flow shop scheduling problems. The algorithm is easily implementable and performs quite effectively. Genetic Algorithms are easy to apply to a wide range of problems, from optimization problems like the traveling salesperson problem, to inductive concept learning, scheduling, and layout problems. The results can be very good on some problems, and rather poor on others. Many scheduling problems are NP-hard problems. For such NP-hard combinatorial optimization problems, heuristics play a major role in searching for near-optimal solutions. If only mutation is used, the algorithm is very slow. Crossover makes the algorithm significantly faster.

In this GA-based heuristic, we generate a different parameter set for the genetic operators. We protect the best schedule which has the minimum makespan, at each generation. Then we transfer this schedule to the next population with no change. This operation enables us to choose the higher crossover and mutation probability $p_{c = 1}$ (crossover probability) and $p_{m = 0.05}$ (mutation probability). So we increase the diversity of the population to get a better solution. We also show the excellent performance of the LOX operator. Most researchers use 0.01 mutation probability in their heuristic. This heuristic uses .05 value of mutation probability and achieved good results.Using a mutation probability higher than 0.05 may reduce the convergence. Investigation of this would lead to a further study of GAs. According to the computational results, the GA-based heuristic success rate is 76% (in Table 2). Therefore, this heuristic is quite effective for flow shop scheduling problems. Also, the GA-based heuristic can be easily extended to solve flow shop problems with other criteria, such as total flow time, maximum tardiness, total tardiness, etc.

10 FUTURE DIRECTION

The future research directions suggested here are intended to bridge the gap between the development of theory and practical applications of theory. Three areas of research are identi-

fied:

- Theoretical,
- Computational,
- Empirical research.

10.1 Theoretical Research

Theoretical research in flowshop scheduling should attempt to develop dominance conditions that are either independent of partial schedules that precede a job candidate or are such that a large number of partial schedules containing a lesser number of jobs are rejected quickly. The dominance conditions developed (in combinatorial and branch and bound procedures) depend on partial schedules that precede a job candidate. theoretical research should consider many more special cases of flowshop scheduling that have been considered before and develop efficient optimization techniques for their solution. Simultaneously, more quick, perhaps dirty but reliable heuristic procedures should be developed. Consideration of hybrid heuristic approaches for these problems provides another fruitful area for future theoretical research.

10.2 Computational Research.

A practical scheduler has difficulty in selecting an algorithm to solve a given flowshop scheduling problem. The computational research should consider such aspects as comparative efficiency of various algorithms for a specified problem with given data set. Thus, new measures of computational effort required should be developed.

In addition, artificial intelligence techniques, such as neural networks should be further exploited to select specific heuristics to be used for a given problem (see Gupta et al. [28] for one such effort).

10.3 Empirical Research

Future research in flowshop scheduling should be inspired more by real life problems rather than problems encountered in mathematical abstractions. For a realistic problem formulation, empirical research is necessary to understand the practical situations. The flowshop scheduling is only one of a few areas where no case histories are available. Empirical research should answer such questions as: What is the maximum problem size encountered in practice? What specific situations give rise to flowshop scheduling problems? What are the desired objectives of scheduling? What is the nature of processing times? How rigid (or flexible) are the operating policies? Empirical research, therefore, needs to include a survey of industrial scheduling practices and situations. Without such a survey, we may in fact spend another twenty-five years in solving a problem that perhaps needs no solution, since it may be the wrong problem (from practical consideration).

We believe study remains to be done in the following areas: It is a good idea to use flow shop specific techniques to speed up the convergence of the algorithm. Algorithms can be developed to optimize population size, number of generations and percentages of genetic operators. New genetic operators can be developed to increase the evolution and convergence speed The recent developments in supply chain management, internet, and e-commerce have created new and complex scheduling and coordination problems that we have just begun to understand. Therefore, we need to diversify our research efforts in scheduling to include these new and emerging problems.

References

[1] B. R. Fox & M. B. McMahon, “Genetic operations for sequencing problems, foundations of genetic algorithms”, *In G. J. E. Rawlins (Ed.)*, San Mateo: Morgan Kaufmann Publishers, pp. 284–300, 1991.

[2] C. Cotta,J. M. Troya, “ Genetic forma recombination in permutation flowshop problems”; *Evol. Comput*., vol. 6, no.1, pp. 25–44, 1998.

[3] C. Koulamas, “A new constructive heuristic for the flowshop scheduling problem”, *Eur J Opl Res*, vol. 105, pp. 66-71, 1998.

[4] C. L.Chen, V.S. Vempati and N. Aljaber N, “An application of genetic algorithms for flow shop problems,” *Eur J Opl Res*, vol. 80, pp. 389-396, 1995.

[5] C. R. Reeves and T. Yamada, “Genetic algorithms, path relinking, and the flowshop sequencing problem”, *Evolutionary Computation*, vol. 6, no. 1, pp. 45–60, 1998.

[6] C. R. Reeves, "A Genetic Algorithm for Flowshop Sequencing", *Computers & Operations Research*, vol. 22, no. 1, pp. 5-13, 1995.

[7] C. R. Reeves, “Improving the efficiency of tabu search for machine scheduling problems”, 1993.

[8] C. Rajendran and H. Ziegler, "Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs", *European Journal of Operational Research*, vol. 155, no. 2, pp. 426-438, 2004.

[9] D. Dannenbring, “An evaluation of flow shop sequencing heuristics”, *Mngt Sci*, vol. 23, pp. 1174-1182, 1977.

[10] D. E. Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, (Addison Wesley: Reading, MA), 1989.

[11] D. S. Palmer, "Sequencing jobs through a multi-stage process in the minimum total time: A quick method of obtaining a near optimum", *Operational Research Quarterly*, vol. 16, no. 1, pp. 101-107, 1965.

[12] D. Whitley, T. Starkwether and D. Shaner, “The travelling salesman and sequence scheduling problems: quality solutions using genetice dge recombination”, *In: Davis L (ed). Hand book of Genetic Algorithms. Van Nostrand Reinhold*, New York, USA, pp 350-372, 1991.

[13] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flow-shop problem", *European Journal of Operational Research*, vol. 91, no. 1, pp. 160-175, 1996.

[14] E.Vallada and R. Ruiz, "Cooperative metaheuristics for the permutation flowshop scheduling problem", *European Journal of Operational Research*, vol. 193, no. 2, pp. :365-376, 2009.

[15] E. Vallada, R. Ruiz and G. Minella, "Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics", *Computers & Operations Research*, vol. 35, no. 4, pp. 1350-1373, 2008.

[16] F. A. Ogbu and D. K. Smith, “The application of the simulated annealing algorithms to the solution of the n/m/Cmax flowshop problem”, *Computers & Operations Research*, vol. 17, no. 3, pp. 243–253, 1990.

[17] F. Werner, “On the heuristic solution of the permutation flowshop problem by path algorithms”, *Journal of the Operational Research Society*, vol. 44, no. 4, pp. 375–382, 1993. Computers & Operations Research, vol. 20, no. 7, pp. 707–722.

[18] G. C. Onwubolu and D. Davendra, "Scheduling flow shops using differential evolution algorithm", *European Journal of Operational Research*, pp. 171, no. 2, pp. 674-692, 2006.

[19] G. Minella, R. Ruiz and M. Ciavotta, "A review and evaluation of multiobjective algorithms for the flowshop scheduling problem", *INFORMS Journal on Computing*, vol. 20, no. 3, pp. 451-471, 2008.

[20] H. Davoud Pour, “ A new heuristic for the n-job, m-machine flow-shop problem”, *Production Planning and Control*, vol.12, no.7, pp. 648–653, 2001.

[21] H. G. Campbell, R. A. Dudek and M. L. Smith, “Heuristic algorithm for N-job, M-machine sequencing problem”, *Management Science Series B-Application*, vol. 16, no. 10, pp. B630-B637, 1970.

[22] H. Ishibuchi, N. Yamamoto, T. Murata and H. Tanaka, “Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems”, *Fuzzy Sets and Systems*, vol. 67, pp. 81–100, 1994.

[23] I. H. Osman and C. N. Potts, "Simulated Annealing for Permutation Flow-shop Scheduling", *Omega-International Journal of Management Science*, vol. 17, no. 6, pp. 551-557, 1989.

[24] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion", *Computers & Operations Research*, vol. 31, no. 1, pp. 1891-1909, 2004.

[25] J. H. Holland, “Adaptation in Natural and Artificial Systems”, *University of Michigan Press*, Ann Arbor, USA, 1975.

[26] J. J. Grefenstette, R. Gopal B. Rosmaita and D. Van Gucht, “ Genetic algorithms for the travelling salesman problem”, *In: Grenfenstette JJ (ed). Proceeding of the First International Conference on Genetic Algorithms and Their Applications.* Carnegie-Mellon University, Pittsburgh, P A, USA, pp 160-168, 1985.

[27] J. M. Framinan and R. Leisten, "An efficient constructive heuristic for flowtime minimization in permutation flow shops, Omega, vol. 31, pp. 311-317, 2003.

[28] J. N. D. Gupta, “A functional heuristic algorithm for the flow-shop scheduling problem”, *Opl Res Q*, vol. 22, pp. 39-47, 1971.

[29] K. Premalatha and A. M. Natarajan, “Hybrid PSO and GA for global maximization”, *International Journal of Open Problems in Computer Science and Mathematics*, Vol. 2, No. 4. pp. 597-608, 2009.

[30] L. Davis, “Job shop scheduling with genetic algorithms”, *In: Grenfenstette JJ (ed).* Proceedings of the First International Conference on Genetic Algorithms. Carnegie-MellonU niversity, Pittsburgh, PA, USA, pp 136-140, 1985.

[31] L. Wang and D. Z. Zheng, “An effective hybrid heuristic for flow shop scheduling”, *The Inter. J. Advanc. Manufact. Tech*, vol. 21, pp. 38–44, 2003a.

[32] L. Wang, L. Zhang, “Determining optimal combination of genetic operators for flow shop scheduling”, *Int. J. Adv. Manuf. Technol*, vol. 30, pp. 302–308, 2006.

[33] L. Wang, L. Zhang, D. Z. Zheng, “A class of order-based genetic algorithm for flow shop scheduling”, *Int. J. Adv. Manuf. Technol*, vol. 22, pp. 828–835, 2003.

[34] L. Wang, L. Zhang, D. Z. Zheng, “The ordinal optimisation of genetic control parameters for flow shop scheduling”, *Int. J. Adv. Manuf. Technol*, vol. 23, pp. 812–819, 2004.

[35] L. Zhang, L. Wang, D. Z. Zheng, “An adaptive genetic algorithm with multiple operators for flowshop scheduling”, *Int. J. Adv. Manuf. Technol*, vol. 27, pp. 580–587, 2006.

[36] M. F. Tasgetiren, Y. C. Liang, M. Sevkli and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem", *European Journal of Operational Research*, vol. 177, no. 3, pp. 1930-1947, 2007.

[37] M. Gen and R. Cheng, “Genetic algorithms and engineering optimization”, *John Wiley*, *New York*, 2000.

[38] M. Ibrahim, Alharkan, “Algotithms for Sequencing and Scheduling”, *Industrial Engineering King Saud University Riyadh*, Saudi Arabia.

[39] M. Nawaz, Jr. E. E. Enscore and I. Ham, "A heuristic algorithm for the m machine, n job flowshop sequencing problem", *Omega-International Journal of Management Science*, vol. 11, no. 1, pp. 91-95, 1983

[40] M. R. Garey, D. S. Johnson and R. Sethi, "The complexity of flowshop and jobshop scheduling", *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117-129, 1976.

[41] M. S. Nagano and J. V. Moccellin, “A high quality solution constructive heuristic for flow shop sequencing”, *Journal of the Operational Research Society*, vol. 53, pp. 1374–1379, 2002.

[42] P. J. Kalczynski and J. Kamburowski, "An empirical analysis of the optimality rate of flow shop heuristics", *In press at European Journal of Operational Research*, 2009.

[43] P. J. Kalczynski and J. Kamburowski, “On the NEH heuristic for minimizing the makespan in permutation flow shops”, OMEGA, *The International Journal of Management Science*, vol. 35, no.1, pp. 53–60, 2007.

[44] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics", *European Journal of Operational Research*, vol. 165, no. 2, pp. 479-494, 2005.

[45] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem", *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033-2049, 2007.

[46] R. Ruiz, C. Maroto and J. Alcaraz, "Two new robust genetic algorithms for the flowshop scheduling problem", *Omega-International Journal of Management Science*, vol. 34, no. 5, pp. 461-476, 2006.

[47] S. F. Rad, R. Ruiz, R. and N. Boroojerdian, "New high performing heuristics for minimizing makespan in permutation flowshops", *Omega-International Journal of Management Science*, vol. 37, no. 2, pp. 331-345, 2009.

[48] S. K. Iyer, B. Saxena, “Improved genetic algorithm for the permutation flowshop scheduling problem”, *Comput. Oper. Res*, vol. 31, no. 4, pp. 593–606, 2004.

[49] S. M. Johnson, “Optimal two- and three-stage production schedules with setup times included”, *Naval Research logistics Quarterly*, vol. 1, pp. 61–68, 1954.

[50] T. Murata,H. Ishibuchi and H. Tanaka, “Algorithms for flowshop scheduling problems”, *Comput. Indust. Eng*, vol. 30, pp. 1061–1701, 1996.

[51] T. Stuttzle, “Applying iterated local search to the permutation flow shop problem”, *Technical report*, AIDA-98-04, TU Darmstadt, FG Intellektik, 1998.

[52] X. Y. Dong, H. K. Huang and P. Chen, "An improved NEH-based heuristic for the permutation flowshop problem", *Computers & Operations Research*, vol. 35, no.12, pp. 3962-3968, 2008.