A Hybrid Evolutionary Algorithm for Solving Flexible Job Shop Scheduling Problem

Mahmoud Riad Mahmoud
Institute of Statistical Studies & Research

Mohamed Sayed Ali Osman,
Higher Technological Institute

Ramadan Abd El-hamed Zean El-Deen
Institute of Statistical Studies & Research

Abd Al-azeem Mohamed Abd Al-azeem
Institute of Statistical Studies & Research

Abstract *This paper presents an Evolutionary Algorithm (EA) to solve the flexible job shop scheduling problem, especially minimizing the makespan. A hybrid algorithm is introduced for solving flexible job shop scheduling problem. The proposed algorithm consists of three sequential stages. The first stage is a new technique for initializing feasible solutions, is used as initial population for the second stage. The second stage uses genetic algorithm to improve the solutions that have been found in the first stage. The final stage uses tabu search to improve the best solution that has been found by genetic algorithm. The Job Shop Scheduling Problem (JSSP) is an NP-hard combinatorial optimization problem that has long challenged researchers. A schedule is a mapping of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. One of the objectives of the JSSP is to find a schedule that minimizes the makespan [12]. Some problems from references are solved using the proposed algorithm and an implementation study is presented. The implementation study shows the efficiency of the proposed algorithm.*

Key words: *flexible job shop scheduling, genetic algorithm, heuristic and tabu search*

1. Introduction

In the classical Job Shop Scheduling Problem (JSSP), n jobs are processed to completion on m unrelated machines. Each job requires processing on each machine exactly once. For each job, technology constraints specify a complete, distinct routing, which is fixed and known in advance. Processing times are sequence independent, fixed, and known in advance. Each machine is continuously available from time zero, and operations are processed without interruption (non-preemption). The common objective is to minimize the maximum completion time (makespan). Each machine is limited to executing one operation (one step of the job route) at a time. Each job has a release-date (the time after which the operations in the job may be executed) and a due-date (the time by which the last activity in the job must finish).

Scheduling has been defined as "the art of assigning resources to tasks in order to insure the termination of these tasks in a reasonable amount of time", or it is allocation of resource over time to perform a collection of tasks [9].

From the area of complexity theory we know that problems can be divided into different classes to enable a differentiation concerning the effort (runtime or memory) that is required to complete a globally best solution [13]:

1. The class of P-problems contains all problems for which the effort in relation to the problem dimension can be dominated by a polynomial function. Consequently the best solution for such problems can be found in most cases due to the computing power available today. It is even possible in the worst case just to try all possible solution candidates iteratively.
2. The class of NP-problems in contrast represents such instances whose effort increases more than polynomial (i.e. exponential or even over-exponential) depending on the problem size. Therefore, problems of that type cannot be solved algorithmically even in rather low dimensions because of the enormous number of potential solution candidates that makes it impossible to find the global optimum in acceptable time.

Job Shop scheduling is a well known scheduling problem. It is NP-hard and one of the most intractable combinatorial problems [3]. There are many methods to solve scheduling problems. These methods are trying to find optimal solution (exact solution) or acceptable solution (approximate solution). In most cases, to find the optimal solution is unpractical (some times impossible). Since it's impossible to find the optimal solution, it's necessary to use methods that find a good solution but no guarantee for how far that solution from the optimal solution. For example, tabu search and genetic algorithms are methods that are used to find an approximate solution.

Tabu search methods have been applied successfully for the scheduling problems and as solvers of mixed integer programming problems [9]. Nowicki and Smutnicki [7] implemented tabu search methods for job shop and flow shop

scheduling problems. Vaessens [7] showed that tabu search methods (in specific job shop scheduling cases) are superior over other approaches such as simulated annealing, genetic algorithms, and neural networks.

The basic idea of tabu search is to explore the search space of all feasible scheduling solutions by a sequence of moves. A move from one schedule to another schedule is made by evaluating all candidates and choosing the best available, just like gradient-based techniques. Some moves are classified as tabu (i.e., they are forbidden) because they either trap the search at a local optimum, or they lead to cycling (repeating part of the search). These moves are put onto something called the tabu List, which is built up from the history of moves used during the search. These tabu moves force examination of the search space until the old solution area (e.g., local optimum) is left behind. Another key element is that of freeing the search by a short-term memory function that provides "strategic forgetting". Tabu search methods have been evolving to more advanced frameworks that include longer term memory mechanisms. These advanced frameworks are sometimes referred as Adaptive Memory Programming (AMP) [9].

Genetic Algorithms (GAs) are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. As you can guess, genetic algorithms are inspired by Darwin's theory of evolution. When solving a problem using GA, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called search space (also state space). Each point in the search space represents one possible solution. Each possible solution can be "marked" by its value (or fitness) for the problem. With GAs, we look for the best solution among a number of possible solutions where the best solution represented by one point in the search space.

The next section is assigned to the problem description, and then followed by section three which is assigned to describe the proposed hybrid algorithm. In addition, section four is assigned to give an illustrative example on the proposed algorithm, and in section five, the implementation part is given. Finally, section six represents the conclusion.

2. Description of flexible job shop scheduling

Flexible job shop is a generalization of the job shop and the parallel machine environment, which provides a closer approximation to a wide range of real manufacturing systems [٦].

The flexible job shop problem is more complex than the job shop problem because of the additional need to determine the assignment of operations to machines.

The total flexible job shop scheduling problem can be described as:

1. A set of *n* jobs = $J = \{j_1, j_2, ..., j_n\}$ and these jobs are independent of one another. In addition, each job j_r (r = 1, 2, ..., n) is available from time zero. And, there is no priority for achieving any job earlier than others.
2. Each job consists of a set of *x* operations = $\{o_{i\,r}\}_{i=1}^{x}$ where $o_{i\,r}$ means operation number i of a job j_r.
3. These operations should be achieved in a predetermined order $x_{o\,r}$ which means the order of operation number o of the job j_r
4. Each operation should be assigned for a determined machine from a set of m machines = $M = \{M_1, M_2, ..., M_m\}$
5. An operation which has started can't leave the machine until completion (non-preemption condition).
6. Each machine can perform operations one after another.
7. Each machine has the capability to achieve any operation.
8. The time required to complete the whole jobs constitutes the makespan $C_{\max}$ ($C_{\max} = \max \{C_r\}_{r=1}^{n}$, and C_r = completion time of all operation of job j_r).
9. The objective is to determine the set of completion times for each operation which minimizes $C_{\max}$.

Of course, for simplicity requirements, using some notations are essential for making a mathematical model of the total flexible job shop scheduling problem as follows:

- $o_{i\,r}$ means operation number i of job j_r
- $x_{o\,r}$ means the order of operation number o of job j_r
- $P_{x\,r\,q}$ = processing time of operation with order x of job j_r on machine M_q

- $s_{x\,r\,q}$ = starting time of operation with order x of job j_r on machine M_q
- C_r = completion time of all operation of job j_r
- $I_{x\,q} = \begin{cases} 1, & \text{if machine } M_q \text{ is assigned for} \\ & \text{the operation with order x} \\ 0, & \text{otherwise} \end{cases}$
- $x_{t,r}$ = the total number of operations of a job j_r
- $m_{t,r}$ = the total number of machines available for the job j_r
  - MR_b = the time that machine b (M_b) is available to work.
  - n is the total number of jobs.
  - The objective is to determine the set of completion times for each operation which minimizes $C_{\max}$.

So, formulating a mathematical model for the total flexible job shop scheduling problem can be constructed as follows:

$$\text{Min } [z = \max \{C_r\}_{r=1}^{n}], \text{ where } C_r = \sum_{x=1}^{x=x_{t,r}} \sum_{q=1}^{q=m_{t,r}} I_{x\,q}\ P_{x\,r\,q}$$

Subject to:

$$s_{(x+1)rb} = \max \{ s_{x\,r\,a} + P_{x\,r\,a},\ MR_b \}, \quad \forall M_a, M_b \in M$$

$$s_{1rq} \in [0, \infty[\quad , \forall r, q$$

$$MR_b \in [0, \infty[\quad , \forall b$$

$$C_j > 0$$

$$p_{x\,j\,m} > 0$$

$$I \in \{0, 1\}$$

Bruker and Schlie [٦] were among the first to address the Flexible job shop problem. They developed a polynomial algorithm for solving the flexible job shop scheduling problem with two jobs. Chambers [٦] developed a tabu search algorithm to solve the problem. Mastrolilli and Gambardella [٦] proposed two neighborhood functions for the Flexible job shop problem. Yang [٦] presented a new genetic algorithm (GA)-based discrete dynamic programming approach. Kacem and Borne [٦] proposed the approach by localization to solve the resource assignment problem, and an evolutionary approach controlled by the assignment model for the Flexible job shop problem. Wu and Weng [٦] considered the problem with job earliness and tardiness objectives, and proposed a multiagent scheduling method. Xia and Wu [14] treated this problem with a hybrid of particle swarm optimization and simulated annealing as a local search algorithm. Zhang and Gen [16] proposed a multistage operation-based genetic algorithm to deal with the Flexible job shop problem from a point view of dynamic programming.

3. Description of evolutionary algorithm

Evolutionary Algorithms are in the area of study which uses the principles of Darwin's evolution to search large number of feasible solutions for finding the optimal (best) solution in complex problems [5].

One of the extra benefits of evolutionary algorithms over standard search or optimization techniques is the algorithm's modularity. For each of the stages in an evolutionary algorithm, whether creating the initial sample of candidate solutions, called chromosomes, scoring a generation or creating a new generation, the same (or similar) task is done on every chromosome in the sample. This allows evolutionary algorithms to be multithreaded easily so that they can be run in parallel. The ways that this basic algorithm can be implemented are many. The earliest and most basic algorithms based on the above are the Genetic Algorithm (GA) that was pioneered by John Holland from the 1960s and Genetic Programming in the early 1970s [5].

The proposed evolutionary algorithm is a hybrid algorithm, consists of three algorithms (stages):

1. Dispatching rule (shortest processing time SPT)

2. Genetic algorithm
3. Tabu search

In the beginning, the proposed algorithm starts by creating initial solutions (initial population of genetic algorithm stage) using the principle of SPT. Next, genetic algorithm is trying to improve the result of the SPT stage (initial solutions). Finally, tabu search algorithm stage is used to improve the solution that is found by the genetic algorithm stage.

The big challenge was to find a suitable representation for encoding a solution that make it is easy to move from one stage (algorithm) to another. Thinking that, the most difficult part was in the genetic algorithm stage and how to find that representation that could deal with genetics' steps. So, genetic algorithm and chromosome representation has been focused in for finding a suitable way to encode the solutions of the problem.

3.1. Encoding the problem for genetic algorithm

Generally, chromosomes are encoding as simple binary vectors. Unfortunately, this approach cannot frequently be used for real world engineering problems such as combinatorial ones. Two categories of encoding are used for real world engineering problems [١١]:

- Direct chromosome representation.
- Indirect chromosome representation.

The first, the direct chromosome representation can represent a scheduling problem by using the schedule itself as a chromosome. This method generally requires developing specific genetic operators. The second, the indirect chromosome representation does not directly represent a schedule, and transition from the chromosome representation to a legal schedule builder (decoder) is needed prior to evaluation. In the proposed algorithm, direct representation was chosen to give feasibility and legibility to a chromosome and simplicity of utilization for a user. There are two direct representational chromosomes:

1. Parallel Machines Encoding (PME).
2. Parallel Jobs Encoding (PJE).

The second type of direct chromosomes representation was used. It is a direct encoding which permits to solve some of the problems met in the first encoding (indirect representation) such as illegal schedules (they can't be understood without a decoder) after a crossover operation and the creation of the first population. Indeed, this encoding integrates the precedence constraints.

3.2. Shortest processing time algorithm

Although dispatching rules are not better than the local search methods, they are the more frequently applied heuristics due to their simplicity of implementation and their low time complexity. When a machine is available, a priority-based dispatching rule examines the waiting jobs and selects the job with the highest priority to be processed next. Recently, the introduction of composite dispatching rules (CDRs) has been increasingly investigated by the some researchers. These rules are the heuristic combination of single dispatching rules that aim to take over the advantages of the former. The results show that, with careful combination, the composite dispatching rules do perform better than the single ones in the quality of schedules [١٨].

Depending on the specification of each rule, it can be classified [١٨] into:

- Simple Priority Rules
- CDRs
- Weighted Priority Indexes
- Heuristic Scheduling Rules

Simple Priority Rules (SPRs) are usually based on a single objective function. They usually involve only one model parameter, such as:

- Processing time
- Due date
- Number of operations or arrival time

The Shortest Processing Time (SPT) is an example of a SPR. It orders the jobs on the queue in the order of increasing processing times. When a machine is freed, the next job with the shortest time in the queue will be removed for processing. SPT has been found to be the best rule for minimizing the mean flow time and number of tardy jobs.

The CDRs have been studied to join good features from such SPRs. There are two kinds of CDRs presented in literature:

- The first type involves arranging a select number of SPRs at different machines or work centers. Each machine or work center uses a single rule. When a job enters a specific machine, it is processed by the SPR that is predetermined for that machine.
- The second type involves applying the composition of several SPRs to evaluate the priorities of jobs on the queue. The latter type is executed similarly to SPRs; when a machine is free, this CDR evaluates the queue and then selects the job with the highest priority.

Weighted priority index rules are the linear combination of SPRs described above with computed weights. Depending on specific business domains, the importance of a job determines its weight.

Heuristic rules are rules that depend on the configuration of the system. These rules are usually used together with previous rules, such as SPRs, CDRs or weighted priority index rules.

The proposed algorithm uses the principal of a Simple Priority Rule (shortest processing time) to initialize some feasible solutions. The initialized solutions will be used as the initial population of the genetic algorithm stage. Using SPT as a first stage gives the algorithm a better start than initializing the initial solution randomly.

The first stage, the SPT algorithm is used for getting solutions that will be used as initial population for the genetic algorithm is described as followed:

- Step 1. Scan for unscheduled jobs.
- Step 2. Select randomly one job (suppose the job is b)
- Step 3.
  For j = 1 to x (number of operations of b)
  Assign a machine with the smallest processing time to achieve x_j (select a machine randomly if there are more than one machine have the smallest processing time).
  j = j + 1
  Next
- Step 4. Go to step 1 until all jobs are scheduled.
- If all operations of each job is assigned for a machine, calculate the start time for each operation column by column (by the sequence column 1, column 2, column 3,…….)

3.3. Genetic algorithm

Genetic Algorithms (GAs) are a general methodology for investigating a discrete solutions space in a manner that is similar to process of natural selection procedure in biological systems.

Genetic algorithm is one of the stochastic search algorithms based on biological evolution. In order to solve a clearly defined problem and an offspring represented the candidate of solutions. GA is according to crossover and mutation operators with their probabilities to produce a set of offspring chromosomes. GA likes an over and over process, and tries to find one or more highly fit chromosomes [4].

Genetic algorithms, as the name implies, are a type of algorithms, not a single one. This means that many variants of the basic idea exist, and that individual applications may be highly different. However, every variant should include the following operations [15]:

1. A method for encoding solutions to the problem into a string of characters.

2. An evaluation function which takes a string as an input and returns a fitness value which measures the quality of the solution the strings describes.
3. An adaptive plan, whose purpose is to produce new, improved generation of solutions from the current one.

The advantage of applying GAs to hard optimization problems lies in their ability to scan broader regions of the solution space than heuristic methods based upon neighborhood search do. Nevertheless, also GAs are frequently faced with a problem which, at least in its impact, is quite similar to the problem of fall into a local optimum. This drawback is called *premature convergence* in the terminology of GAs [1].

3.3.1. Crossover operators

In this section, crossover operator is presented to adapt to the parallel jobs encoding. The algorithm of this crossover is presented as follows:

- Step 1. Choose randomly the best two parents (chromosomes) and select randomly one job (a row of the matrix). Suppose that the job J is randomly selected.
- Step 2. The operations of job J in Child 1 received the same machines as assigned to the same job J of Parent 2.
- Step 3. Browse all of the jobs (the rows) R of parent 1
  While ((1 ≤ R ≤ N) and (R ≠ J)) do
  Copy the remainder of the machines assigned to the operations of job R of Parent 2 in the same job R of Child 1
  R = R + 1
  End
- Step 4. To obtain Child 2, go to Step 2 and interchange the roles of Parent 1 and Parent 2.

3.3.2. Mutation operator

In this part, a mutation operator is presented, and is called the controlled mutation operator. This operator is designed for parallel jobs encoding where it can balance the machine loads.

The algorithm of this operator is presented as follows:

- Step 1. Choose randomly one chromosome S and calculates the load of all machines in the shop according to S.
- Step 2. Choose randomly one operation from the set of operations in chromosome S assigned to a machine with a high load.
- Step 3. Assign this operation to another machine with a small load.

3.3.3. Fitness function

Fitness is a measure of how well an algorithm has learnt to predict the outputs from the inputs. The goal of having a fitness evaluation is to give feedback to the learning algorithm regarding which individuals should have a higher probability of being allowed to multiply and reproduce and which of them should have a higher probability of being removed from the population.

Evaluation functions play the same role in EAs as the environment in natural evolution. It must indicate the aspects of schedules which make them seem right or wrong to the users. The objective is to minimize the makespan.

The makespan $C_{\max}$ is calculated, according to the following algorithm:

Select chromosome X = chromosome 1
For X = 1 to popsize (population size)
Select job L = job 1
While ($1 \le L \le N$) do
Calculate C_L = time of completion of the last scheduling operation of job L
L = L + 1
End While

$C_{\max}$ X = Max $\{C_1, C_2, \ldots, C_N\}$, ($N$ is the number of jobs)
X = X +1
Next

For each chromosome the fitness function aims to find the minimum of all $C_{\max}$, and is represented as follows:
Fitness = Min $\{C_{\max}1, C_{\max}2, \ldots, C_{\max}X\}$, ($X$ is the population size)

3.4. Tabu algorithm

The tabu search was proposed by Glover. TS try to keep track of the visited search space to avoid revisiting the same solutions and thus improving the efficiency of search [2]. So, the proposed algorithm used three types of tabu list, in tabu search algorithm stage. It can be described as follows:

1. Machine-tabu-list: the machine is not being allowed to be assigned until trying all machines.
2. Job-tabu-list: the job is not being allowed to be visited until completing all jobs.
3. Operation-tabu-list: the operation is not being allowed to be selected until completing all operations.

Each step of the proposed tabu search algorithm stage is illustrated in the next example.

4. An illustrative example

Three jobs and five machines are considered. The operating sequences of these jobs are as follows:

- Job1: $o_{1,1} \rightarrow o_{2,1} \rightarrow o_{3,1}$
- Job2: $o_{1,2} \rightarrow o_{2,2} \rightarrow o_{3,2}$
- Job3: $o_{1,3} \rightarrow o_{2,3}$

In which $o_{2,3}$ means operation two in job three. This problem is an example of a total flexibility in which any operation can be performed equally well by any machine with different processing times. According to the machine used, the processing time of operations is different as described in table 1. All machines and jobs are available from time zero. Non-preemptive principle is used. Besides, parallel jobs encoding was used to represent any solution. In this case, for each operation, the proposed algorithm determines the following:

$$(\text{machine number, starting time, processing time,completion time})$$

For example, an operation has $(M_3, 4, 2, 6)$ that means the assigned machine is M_3 and the starting time of the operation on M_3 is 4 and the processing time is 2 and the completion time is 6.

Table 1 Processing time of the operations on different machines.

x_j	operation	M_1	M_2	M_3	M_4	M_5
x_1	1, 1	1	8	3	7	5
x_2	2, 1	3	5	2	6	4
x_3	3, 1	6	7	1	4	3
x_1	1, 2	1	4	5	3	8
x_2	2, 2	2	8	4	9	3
x_3	3, 2	9	5	1	2	4
x_1	1, 3	1	8	9	3	2
x_2	2, 3	5	9	2	5	3

4.1. Initial population

Some symbols were used to make it easy to describe each chromosome as the following:

1. GenNo *i* = Generation number *i*
2. Ch *i* = Chromosome number *i*

As an example, two solutions were initialized by using SPT technique as explained before. The next two chromosomes ((GenNo 0, Ch1) and (GenNo 0, Ch2)) represent the two solutions generated by the SPT stage.

(GenNo 0, Ch1)

J_j	Random order	x_1	x_2	x_3	$C_{\max J}$	$\max C_{\max}$
J_1	1	$(M_1,0,1,1)$	$(M_3,1,2,3)$	$(M_3,5,1,6)$	6	
J_2	3	$(M_1,2,1,3)$	$(M_1,3,2,5)$	$(M_3,6,1,7)$	7	7
J_3	2	$(M_1,1,1,2)$	$(M_3,3,2,5)$		5	

(GenNo 0, Ch2)

J_j	Random order	x_1	x_2	x_3	$C_{\max J}$	$\max C_{\max}$
J_1	2	$(M_1,1,1,2)$	$(M_3,2,2,4)$	$(M_3,7,1,8)$	8	
J_2	1	$(M_1,0,1,1)$	$(M_1,3,2,5)$	$(M_3,6,1,7)$	7	8
J_3	3	$(M_1,2,1,3)$	$(M_3,4,2,6)$		6	

4.2. Genetic algorithm

In the proposed genetic algorithm stage, genetic algorithm used the initial solutions that are generated in the first stage as the initial population.

4.2.1. Crossover steps

- Generating (GenNo 1, Ch1) by Selecting randomly a job (as an example J_1) from (GenNo 0, Ch2) and assign its machines for (GenNo 1, Ch1) to get the same machines assigned
- Complete the remaining machines of (GenNo 1, Ch1) from (GenNo 0, Ch1).
- Generating (GenNo 1, Ch2) is the same steps for generating (GenNo 1, Ch1) by switching Ch1 and Ch2.
- Calculate the starting time for each chromosome by using random order for jobs for each chromosome.

The next two solutions ((GenNo 1, Ch1) and (GenNo 1, Ch2)) represent the two chromosomes after crossover steps.

(GenNo 1, Ch1)

J_j	Random order	x_1	x_2	x_3	$C_{\max J}$	$\max C_{\max}$
J_1	1	$(M_1,0,1,1)$	$(M_3,1,2,3)$	$(M_3,5,1,6)$	6	
J_2	2	$(M_1,1,1,2)$	$(M_1,3,2,5)$	$(M_3,6,1,7)$	7	7
J_3	3	$(M_1,2,1,3)$	$(M_3,3,2,5)$		5	

(GenNo 1, Ch2)

J_j	Random order	x_1	x_2	x_3	$C_{\max J}$	$\max C_{\max}$
J_1	3	$(M_1,2,1,3)$	$(M_3,3,2,5)$	$(M_3,6,1,7)$	7	
J_2	2	$(M_1,1,1,2)$	$(M_1,3,2,5)$	$(M_3,5,1,6)$	6	7
J_3	1	$(M_1,0,1,1)$	$(M_3,1,2,3)$		3	

4.2.2. Mutation steps

- Select (GenNo 1, Ch2) for mutation and calculate machine load for the selected chromosome as illustrated in the next table 2.
-

Table 2 Calculation of machine load of (GenNo. 1, Ch2)

M_1	M_2	M_3	M_4	M_5
5	0	6	0	0

- Select M_3 (the highest load machine) to be exchanged by a machine with the smallest load.
  - Select randomly a job where M_3 is assigned to achieve an operation (suppose J_1).
  - Select randomly an operation of the selected job J_1 (suppose x_3).
  - Select randomly from the machines with the smallest load, a machine for completing x_3.
  - Exchange M_3 by M_5 to achieve operation x_3 of job J_1.

The next chromosome ((GenNo 1, Ch3)) represents the solution after mutation steps.

(GenNo 1, Ch3) J_j	Random order	x_1	x_2	x_3	$C_{\max J}$	$\max C_{\max}$
J_1	1	$(M_1,0,1,1)$	$(M_3,1,2,3)$	$(M_5,3,3,6)$	6	
J_2	2	$(M_1,1,1,2)$	$(M_1,3,2,5)$	$(M_3,5,1,6)$	6	6
J_3	3	$(M_1,2,1,3)$	$(M_3,3,2,5)$		5	

4.2.3. Fitness

In fitness stage, the makespan is calculated for each chromosome to determine the chromosomes that will be survive for the next population as illustrated in the next table 3.

Table 3 Calculate fitness of each chromosome

	(GenNo 0, Ch1)	(GenNo 0, Ch2)	(GenNo 1, Ch1)	(GenNo 1, Ch2)	(GenNo 1, Ch3)
$\max C_{\max}$	7	8	7	7	6

4.2.4. Selection

The next population (population number 1) has the same size of the previous population. Selection step is completed by selecting chromosomes randomly from offspring (chromosomes) which have been created in the previous step (offspring consist of the original population and two chromosomes that created by crossover step, and finally, one chromosome that created by mutation step) that have the smallest $C_{\max}$ (the best fitness).

4.2.5. Next population

In this step, the algorithm Uses population number l (l = 1, 2, 3, ...) to generate next population number (l+1) and so on until the desired number of populations are get.

4.3. Tabu search

After generating the required number of population using the genetic algorithm, Select randomly the solution (chromosome) with the smallest $C_{\max}$ and let us suppose it's as (GenNo. 1, Ch3).

(GenNo 1, Ch3)

J_j	Random order	x_1	x_2	x_3	$C_{\max J}$	$\max C_{\max}$
J_1	1	$(M_1,0,1,1)$	$(M_3,1,2,3)$	$(M_5,3,3,6)$	6	
J_2	2	$(M_1,1,1,2)$	$(M_1,3,2,5)$	$(M_3,5,1,6)$	6	6
J_3	3	$(M_1,2,1,3)$	$(M_3,3,2,5)$		5	

Tabu search algorithm procedures:

1. Select the first job in the chromosome (J_1).
2. Put the selected job in job-tabu-list
3. Select by ordering an available operation of J_1 (for the first time it will be x_1)
4. Put the selected operation (x_1) in operation-tabu-list
5. Put the machine assigned for completing x_1 in a machine-tabu-list (to not be visited until trying other machines available for x_1).
6. Exchange the machine assigned for achieving x_1 by another available machine (say M_a)
7. Calculate the starting time for the new solution (using the same order assigned for the solution before).
8. Calculate $C_{\max}$ for the new solution = $C_{\max S_i}$
9. If the new ($\max C_{\max}$) is smaller than the old make a move.
10. Add M_a in machine-tabu-list
11. Go to step 6 until completing all available machines.
12. Clear machine-tabu-list.
13. Go to step 3 until completing all operations.
14. Clear operation-tabu-list.
15. Go to step 1 until selecting all jobs.
16. Clear job-tabu-list.

If it is decided to use multi-start of tabu, it is done only by exchange step 9 of the tabu algorithm by the following:

1. If the new ($\max C_{\max}$) is smaller than or equal the old make a move.

5. Implementation

The implementation was done on the system of Microsoft, Windows XP Professional Service Pack 2 and on a computer with Intel Pentium III processor 800 MHz and 256 MB of RAM. A special program was coded using visual basic 6 to implement the proposed hybrid algorithm.

5.1. The First Test (10×6 problem)

This problem presents a partial flexibility. The problem consists of 10 jobs and 6 machines and each job has 5 or 6 operations in its operating sequence. The problem is called "mk01" and the best known makespan is equal to 40 units of time [10].

The following table 4 shows the results of ten runs. Some notations are used in all tables that illustrate the results, as "I" stands for the number of initial solutions, "P" stands for the population size and "G" stands for the number of generations.

Table 4 Results of the hybrid algorithm for each stage.

I = 50, P = 10, G = 10

Run	Makespan found by				Makespan
No.	SPT	GA	TS	TS with multi-start = 3	(units of time)
1	48	48	48	48	48
2	49	49	46	43	43
3	48	48	44	44	44
4	48	48	45	44	44
5	48	47	42	39	39
6	48	48	45	44	44
7	49	49	47	44	44
8	48	48	45	44	44
9	46	46	41	38	38
10	48	47	43	40	40

It's clear from the results that the use of multi-start technique of the tabu algorithm has the capability to force the solution to be better than the solution that it is found without using multi-start strategy.

Table 5 represents the best solution has been found by the proposed algorithm. Some symbols are used to make it easer to represent the solution, as follows:

- S, refers to start time of the machine
- F, refers to finish time of the machine
- J, refers to the job that the machine is assigned to achieve an operation of that job
- O, refers to the operation order that the machine is assigned to achieve it

Table 5 The best solution (makespan = 38)

M1				M2				M3				M4				M5				M6			
S	**F**	**J**	**O**	**S**	**F**	**J**	**O**	**S**	**F**	**J**	**O**	**S**	**F**	**J**	**O**	**S**	**F**	**J**	**O**	**S**	**F**	**J**	**O**
0	**1**	**4**	**1**	**0**	**2**	**2**	**1**	**0**	**4**	**1**	**1**	**3**	**5**	**7**	**2**	**0**	**3**	**5**	**1**	**0**	**2**	**10**	**1**
5	**7**	**6**	**2**	**2**	**8**	**3**	**1**	**4**	**5**	**2**	**2**	**11**	**14**	**9**	**3**	**6**	**11**	**9**	**2**	**2**	**3**	**7**	**1**
7	**8**	**5**	**2**	**8**	**14**	**4**	**2**	**8**	**12**	**8**	**2**	**14**	**20**	**2**	**4**	**11**	**14**	**1**	**2**	**3**	**5**	**6**	**1**
8	**10**	**2**	**3**	**14**	**20**	**5**	**3**	**12**	**16**	**7**	**3**	**20**	**26**	**10**	**5**	**14**	**17**	**10**	**3**	**5**	**6**	**9**	**1**
16	**17**	**3**	**3**	**20**	**26**	**6**	**4**	**16**	**20**	**6**	**3**	**26**	**32**	**5**	**5**	**17**	**18**	**7**	**4**	**6**	**8**	**8**	**1**
17	**18**	**8**	**3**	**26**	**32**	**8**	**4**	**20**	**21**	**4**	**3**	**32**	**38**	**9**	**6**	**21**	**24**	**4**	**4**	**8**	**14**	**10**	**2**
18	**20**	**9**	**4**	**32**	**38**	**8**	**5**	**21**	**25**	**5**	**4**					**25**	**30**	**3**	**5**	**14**	**16**	**3**	**2**
20	**21**	**1**	**4**					**25**	**26**	**7**	**5**									**16**	**18**	**1**	**3**
26	**27**	**6**	**5**					**26**	**30**	**9**	**5**									**18**	**19**	**10**	**4**
27	**30**	**10**	**6**					**30**	**31**	**1**	**5**									**19**	**25**	**3**	**4**
30	**33**	**6**	**6**					**32**	**36**	**5**	**6**									**25**	**30**	**2**	**5**
																				30	**32**	**4**	**5**
																				32	**38**	**1**	**6**

5.2. The Second Test (10 × 10 problem)

Implementation of the hybrid algorithm was done to a problem with 10 jobs and 10 machines. Each job has 3 operations in its operating sequence. This problem presents a total flexibility [14] [11].

Ten runs were performed and the following table 6 shows the results by using the proposed algorithm for illustrating the efficiency of the proposed hybrid algorithm.]

Table 6 Results of our algorithm for each stage.

I = 50, P = 10, G = 10

Run No.	**Makespan found by**				**Makespan (units of time)**
	SPT	**GA**	**TS**	**TS with multi-start = 3**	
1	**8**	**8**	**8**	**8**	**8**
2	**9**	**8**	**8**	**8**	**8**
3	**9**	**9**	**8**	**8**	**8**
4	**8**	**8**	**8**	**8**	**8**
5	**9**	**9**	**8**	**8**	**8**
6	**9**	**8**	**8**	**8**	**8**
7	**8**	**8**	**8**	**7**	**7**

Table 6 (continued) Results of our algorithm for each stage.

Run No.	**Makespan found by**				**Makespan (units of time)**
	SPT	**GA**	**TS**	**TS with multi-start = 3**	
8	**8**	**8**	**8**	**8**	**8**
9	**9**	**8**	**8**	**8**	**8**
10	**9**	**9**	**9**	**8**	**8**

Table 6 shows the efficiency of our hybrid algorithm and make it clear to think about the SPT algorithm as a good solo technique for solving the flexible job shop scheduling in some cases.

Table 7 gives a comparison between different algorithms that have been applied for solving this problem (10 × 10 problem). In table 5.2.2 and other tables of the comparison between different algorithms, the first column is labeled "Temporal decomposition", refers to F. Chetouane's method. The next column labeled "Classic GA" refers to classical genetic algorithm. The third and forth columns are labeled "Approach by localization" and "AL+CGA", respectively, are two algorithms by Kacem et al. The fifth column is labeled "PSO+SA", refers to the proposed algorithm by Xia and

Wu [14] [6]. The sixth column is labeled "hGA", refers to the hybrid genetic algorithm proposed by Gao, J. et al. [6]. Finally, the last column is labeled "Proposed algorithm", refers to the proposed hybrid evolutionary algorithm.

Table 7 Comparison of results on problem 10×10 with 30 operations

Temporal decomposition	Classic GA	Approach by localization	AL + CGA	PSO + SA	hGA	Proposed algorithm
16	7	8	7	7	7	7

Besides, the success of the proposed hybrid algorithm for finding the best known makespan for this problem, it takes about two minutes to get a solution, which makes the proposed algorithm is the fastest. It takes from the proposed algorithm to find the best solution about 2 minutes, while the "hGA" algorithm takes about 17 minutes to find the best solution [6].

5.3. The Third Test (8 × 8 problem)

This is an instance of partial flexibility. In this flexible job shop problem, there are 8 jobs with 27 operations to be performed on 8 machines [14].

Ten runs were performed and table 8 shows the results by using the proposed algorithm for illustrating the efficiency of the proposed hybrid algorithm.

Table 9 gives a comparison between different algorithms that have been applied for solving this problem (8 × 8 problem).

Table 8 Results of our algorithm for each stage.

I = 50, P = 10, G = 10

Run No.	Makespan found by SPT	GA	TS	TS with multi-start = 3	Makespan (units of time)
1	16	16	16	15	15
2	16	16	16	16	16
3	16	16	16	16	16
4	16	16	15	15	15
5	16	16	15	15	15
6	16	16	16	16	16
7	16	16	16	16	16
8	16	16	16	16	16
9	16	16	16	16	16
10	16	16	16	16	16

Table 9 Comparison of results on problem 8×8 with 27 operations

Temporal decomposition	Classic GA	Approach by localization	AL + CGA		PSO + SA		hGA	Proposed algorithm
19	16	16	15	16	15	16	15	15

It takes from the proposed algorithm to find the best solution about 1.5 minutes, while the "hGA" algorithm takes about 5 minutes to find the best solution [6].

5.4. The Forth Test (15 × 10 problem)

A larger-sized problem is chosen to test the performance of the proposed hybrid evolutionary algorithm. This instance has 15 jobs with 56 operations that have to be processed on 10 machines with total flexibility [14].

Ten runs were performed and table 10 shows the results that have been found by using the proposed algorithm.

Table 11 gives a comparison between different algorithms that have been applied for solving this problem (15 × 10 problem).

Table 10 Results of our algorithm for each stage.

I = 50, P = 10, G = 10

Run No.	Makespan found by SPT	GA	TS	TS with multi-start	Makespan (units of time)
1	15	14	14	14	14
2	15	15	15	15	15
3	14	14	14	14	14
4	15	14	13	13	13
5	14	14	14	14	14
6	15	15	15	15	15
7	14	14	14	14	14
8	15	13	13	13	13
9	15	15	15	15	15
10	15	15	15	15	15

Table 11 Comparison of results on problem 15×10 with 56 operations

Classic GA	AL + CGA	PSO + SA	hGA	Proposed algorithm
23	24	12	11	13

According to the data that is illustrated in table 11, the proposed algorithm did not find the best known makespan, only for this problem. But, it found a solution near the best known in a fast way. It takes from the proposed algorithm to find the best solution (by the proposed algorithm) about 3 minutes, while the "hGA" algorithm takes about 135.47 minutes to find the best solution [6].

5.5. The Fifth Test

In the next section, studying the effects of changing the parameters on the behavior of our algorithm was done. The problem of the second test (10 × 10 total flexible job shop scheduling) was chosen to be solved under various values of parameters. In each test, changing the value of only one parameter and making the others unchanging was done. In each test, five runs were made and the best solution is selected to be illustrated in figure 1, figure 2, figure 3, and figure 4.

Figure 1 (Changing in the number of initial solutions)
Best makespan in five runs: G = 10, P = 10, multi-start = 3

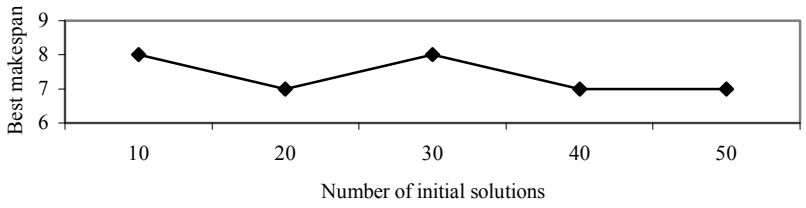


Figure 2 (Changing in the number of generations)
Best makespan in five runs: I = 50, P = 10, multi-start = 3

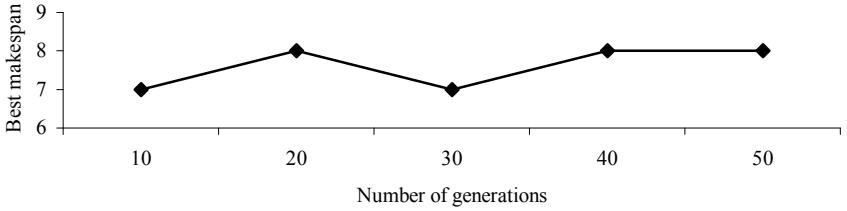


Figure 3 (Changing in the population size)
Best makespan in five runs: I = 50, G = 10, multi-start = 3

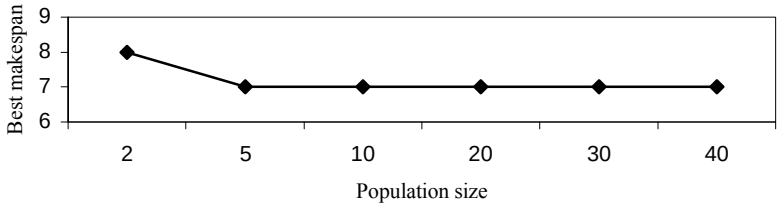
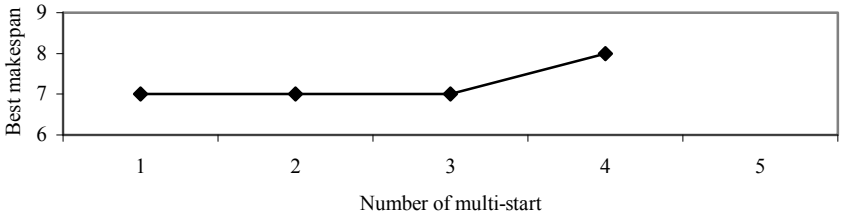


Figure 4 (Changing in the number of multi-start of Tabu)
Best makespan in five runs: I = 50, P = 10, G = 10



6. Conclusion

The application of evolutionary algorithms to a flexible job-shop scheduling problem has been defined. It is demonstrated that choosing a suitable initial solutions and good representation of chromosomes (parallel job encoding) is an important step to get better result in less iterations.

A new method has been developed for generating initial solutions over the chromosome encoding using the concept of SPT which could be considered a solo technique for solving the flexible job shop scheduling problem.

The parameters (number of initial solutions, population size, number of generations, and number of multi-start of tabu algorithm) are selected heuristically.

Implementation results show that the proposed hybrid evolution algorithm is suitable for solving the flexible job shop scheduling problem, confirming the effectiveness of the proposed approach and its fastness.

7. References

[1] Affenzeller, M., Wagner, S. and Winkler, S. (2005), "GA-Selection Revisited from an ES-Driven Point of View". Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Lecture Notes in Computer Science 3562, pp. 262-271. Springer-Verlag, 2005.

[2] Caldeira, J., Melicio, F. and Rosa, A. (2004), "Using a Hybrid Evolutionary-Taboo Algorithm to Solve the JobShop Problem". ACM Symposium on Applied Computing, SAC ’04, March 14-17, 2004, Nicosia, Cyprus.

[3] Caldeira, J., Melicio, F. and Rosa, A. (2005), "Improving on Excellence. An Evolutionary Approach", Proceedings of the 6th WSEAS Int. Conf., on Evolutionary Computing, Lisbon, Portugal, June 16-18, 2005, pp. 15-23.

[4] Chiu, H., Hsieh, K., Tang, Y. and Chien, W. (2007), "A Knowledge-Based Genetic Algorithm for the Job Shop Scheduling Problem", Proceedings of the 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Corfu Island, Greece, February 16-19, 2007.

[5] Crafti, D. (2004). "A Job Shop Scheduler using a Genetic Tree Algorithm". School of Computer Science and Software Engineering, Monash University, Clayton, Victoria, Australia. http://www.csse.monash.edu.au/hons/se-projects/2003/Crafti/thesis.pdf. January 29, 2004.

[6] Gao, J., Gen, M. and Sun, L. (2006), “A Hybrid of Genetic Algorithm and Bottleneck Shifting for Flexible Job Shop Scheduling Problem”, GECCO'06, July 8–12, 2006, Seattle, Washington, USA. Copyright 2006 ACM 1-59593-186-4/06/0007.

[7] Glover, F. (1996), “Tabu search and adaptive memory programming - advances, applications and challenges”. Interfaces in Computer Science and Operations Research. Barr, Helgason and Kennington (eds.), Kluwer Academic Publishers.

[8] Ho, N. and Tay, J. (2005), "Evolving Dispatching Rules for solving the Flexible Job-Shop Problem". Evolutionary Computation, 2005, the 2005 IEEE Congress on 2-5 Sept. 2005, Vol. 3, pp. 2848 – 2855, ISBN: 0-7803-9363-5.

[9] Jones, A. and Rabelo, L. (1998), “Survey of Job Shop Scheduling Techniques”, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD. Journal of Production Research, Vol. 38, pp. 3623-3637.

[10] Mastrolilli, M. and Gambardella, L. (2000), "Effective Neighborhood Functions for the Flexible Job Shop Problem: Appendix". IDSIA - Istituto Dalle Molle di Studi sull'Intelligenza Artificial, C.so Elvezia 36, 6900 Lugano, Switzerland. {monaldo, luca}@idsia.ch, http://www.idsia.ch. February 14, 2000.

[11] Mesghouni, K., Hammadi, S. and Borne, P. (2004), "Evolutionary Algorithms for Job-Shop Scheduling". Int. J. Appl. Math. Comput. Sci., Vol. 14, No. 1, 91–103.

[12] Resende, M. and Ribeiro, C. (2004), “Parallel Greedy Randomized Adaptive Search Procedures”, AT&T Labs Research Technical Report TD-67EKXH. To appear in Parallel Metaheuristics, E. Alba (ed.), John Wiley and Sons, 2005.

[13] Wagner, S., Affenzeller, M. and Schragl, D. (2004), "Traps and Dangers when Modelling Problems for Genetic Algorithms". Austrian Society for Cybernetic Studies. Cybernetics and Systems 2004, pp. 79-84.

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

[15] Zdansky, M. and Pozivil, J. (2002), "Combination Genetic/Tabu Search Algorithm for Hybrid Flowshops Optimization". Proceedings of ALGORITMY 2002, Conference on Scientific Computing, pp. 230-236.

[16] Zhang, H. and Gen, M. (2004), “Multistage-Based Genetic Algorithm for Flexible Job-Shop Scheduling Problem”, the 8th Asia Pacific Symposium on Intelligent and Evolutionary Systems 6th - 7th December 2004. Cairns, Australia, pp. 223-232.