Review Study to Minimize the Make Span Time for Job Shop Scheduling of Manufacturing Industry by Different Optimization Method

Vineet Kumar[1], Dr. Om Pal Singh[2]

[1] *Research Scholar, PTU, Jalandhar*

[2] *Professor, Mechanical Engineering Department. BCET, Gurdaspur*

Abstract: **- Scheduling is one of the most important issues in the planning and operation of manufacturing system, and scheduling has gained much attention increasingly in the recent years. The flexible job shop scheduling problem (JSP) is one of the most difficult problems in this area. It consists of scheduling a set of jobs on a set of machines with the objective to minimize a certain make span time. Each machine is continuously available from time zero, processing one operation at a time without preemption. Each job has a specified processing order on the machine which are fixed and known in advance. Moreover, a processing time is also fixed and known. Different researcher use different algorithms to optimize the make span time. In this paper study has been focused on the different algorithms to optimize the make span time. Now a day's different algorithms that are used are Genetic Algorithm, Artificial Neural Network, Ant Colony Optimization and Particle Swarm Optimization.**

Keywords: **Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Job Shop Scheduling (JSS), Make Span Time.**

I. INTRODUCTION

The job shop scheduling problem is to decide a schedule of jobs that is endowed with pre-set operation series in a multi-machine atmosphere. In the traditional job shop scheduling problem (JSP), n-jobs are processed to the finishing point on m-unrelated machines. For each and every task, technology limitations spell out an absolute and distinctive routing which is set and identified earlier. In addition, processing periods are set and identified previously.

This synopsis deals with the situations in which the effectiveness measure (time, cost, distance, etc.) is a function of the order or schedule of performing a series of jobs (tasks). The selection of the appropriate order in which waiting customers may be served is called scheduling. Scheduling problems can be classified in two groups:

1. In the first group, there are n jobs to be performed, where each job requires processing on some or all of m different machines. The order in which these machines are to be used for processing each job as well as the expected or actual processing time of each job on each of the machines is known. We can also measure the effectiveness for any given schedule of jobs at each of the machines and we wish to select from the $(n!)^m$ theoretically feasible alternativeness measure(e.g., minimizes the total elapsed time from the start of the first job to the completion of the last job as well as idle time of machines). A technologically feasible sequence is one which satisfies the constraints (if any) on the order in which each job must be performed through the m machines. The technology of manufacturing processes renders many schedules technologically infeasible. For example, a part must be degreased before it is painted; similarly, a hole must be drilled before it is threaded.

Although, theoretically, it is always possible to select the best schedule by testing each one; in practice, it is impossible because of the large number of computations involved. For example, if there are 4 jobs to be processed at each of the 5 machines (i.e., n=4 and m=5), the total number of theoretically possible different schedules will be $(4!)^5$ = 7,962,624. Of course, as already said, some of them may not be feasible because the required operations must be performed in a specified order. Obviously, any technique which helps us arrive at an optimal (or at least approximately so) schedule without trying all or most of the possibilities will be quite valuable.

2. The second group of problems deals with job shops having a number of machines and a list of tasks to be performed. Each time a task is completed by a machine, the next task to be started on it has got to be decided. Thus the list of tasks will change as fresh orders are received.

Unfortunately, both types of problems are intrinsically difficult. While solutions are possible for some simple cases of the first type, only some empirical rules have been developed for the second type till now.

In the scheduling problems, there are two or more customers to be served (or jobs to be done) and one or more facilities (machine) available for this purpose. We want to know when each job is to begin and what its due date is. We also want to know which facilities are required to be each job, in which order these facilities are required and how long each operation is to take.

Scheduling problems have been most commonly encountered in production shops where different products are to be processed over various combinations of machines.

However, scheduling problems can arise even where only one service facility is involved, for example, a number of programs waiting to get on a computer or a number of patients waiting for a doctor.

A general scheduling problem may be defined as follows:

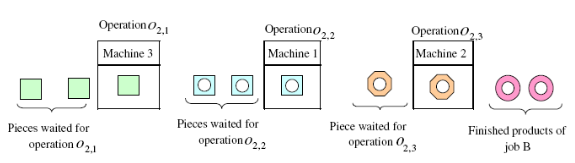
Let there be n jobs (1, 2, 3 ... n), each of which has to be processed, one at a time, on each of m machines (A, B, C ...) The order of processing each job through the machines is given (for example, job 1 is processed on machines A, C, B, in this order). Also, the time required for processing each job on each machine is given. The problem is to find among $(n!)^m$ possible schedules, that technologically feasible schedule for processing the jobs which gives the minimum total elapsed time for all the jobs.

II. DIFFERENT OPTIMIZATION METHOD

In 2009, Parviz Fattahi et al. [1] The literature of FJSP is considerably sparser than the literature of JSP. Bruker and Schile were among the first to address this problem. They developed a polynomial algorithm for solving the flexible job shop problem with two jobs. For solving the realistic case with more than two jobs, two types of approaches have been used: hierarchical approaches and integrated approaches. In hierarchical approaches assignment of operations to machines and the sequencing of operations on the resources or machines are treated separately, i.e. assignment and sequencing are considered independently. In the integrated approaches, assignment and sequencing are not differentiated. Hierarchical approaches are based on the idea of decomposing the original problem in order to reduce its complexity. This type of approach is natural for FJSP since the routing and the scheduling sub-problem can be separated. Brandimarte was the first in applying this decomposition approach for the FJSP. He solved the routing sub-problem using some existing dispatching rules and then focused on the scheduling sub-problem, which is solved using a tabu search heuristic. Saidi and Fattahi presented a mathematical model and a tabu search algorithm to solve the flexible job shop scheduling problem with sequence-dependent setups. They used a hierarchical approach with two heuristic to solve this problem. The first one for assigning each operation to a machine out of a set of capable machines and the second one for sequencing the assigned operations on all machines in order to obtain a feasible schedule minimizing the Makespan. Another work in this field was represented by Kacem et al. and Xia and Wu. Integrated approaches were used by considering assignment scheduling at the same time. Hurink et al. proposed a tabu search heuristic in which reassignment and rescheduling are considered as two different types of moves. The integrated approach which had been represented by Dauzere-Peres and Paulli was defined a neighborhood structure for the problem where there was no distinction between reassigning and resequencing an operation. Mastrololli and Gambardella improved Dauzere-Peres tabu search techniques and presented two neighborhood functions.

This paper considers flexible jobs scheduling problem with overlapping in operations. Since the problem is well known as NP-Hard class, a simulated annealing algorithm is developed to solve large scale problems. Moreover, a mixed integer linear

P. Fattahi et al./Applied Mathematical Modelling 33 (2009) 3076–3087

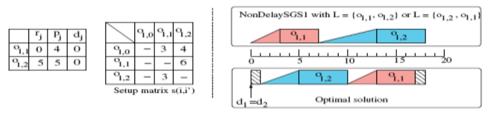

Operation $O_{2,1}$
Operation $O_{2,2}$
Operation $O_{2,3}$
Machine 3
Machine 1
Machine 2
Pieces waited for operation $O_{2,1}$
Pieces waited for operation $O_{2,2}$
Piece waited for operation $O_{2,3}$
Finished products of job B

An illustration of overlapping in operations.

programming (MILP) method is presented to validate the proposed algorithm. The approach is tested on a set of random generated test problems to evaluate the behaviour of the proposed algorithm. The reminder of this paper is organized as follows: Section 2 describes the problem under consideration and presents a mixed integer linear programming model. The solution procedure and hierarchical approach are presented in Section 3. Section 4 presents numerical experiments and discussion. Section 5 includes concluding remarks.

In 2012, D. Prot et al. [2] The tabu search (see Glover & Laguna, 1997) is based on a neighborhood search procedure to iteratively move from a current solution s to a solution s' in the neigborhood N(s) of s until some stopping criterion. In the current study, a solution is represented by a priority list L that contains each operation to schedule with an associate given mode. A direct neighbor then corresponds to a list L0 that can be obtained by doing only one change in the list L. At the first iteration of the search, the first priority list is built using one of the priority rules presented before. Once a list is fixed, the corresponding schedule can be built either using the Semi Active SGS or the NonDelaySGS1. So, we propose two different versions of the tabu search, that we will compare in the final results (see Section 7). This approach is motivated by the fact that, given a priority list, the two SGS do not have the same behavior, and it is easier to manually guide solutions obtained by SemiActiveSGS since the operation order on each machine is exactly the same as in the priority list. However, NonDelaySGS1 implicitly tries to reduce setup times and hence may obtain better results.

D. Prot, O. Bellenguez-Morineau/Computers & Industrial Engineering 62 (2012) 1109–1118



	r_j	p_j	d_j
$o_{1,1}$	0	4	0
$o_{1,2}$	5	5	0

	$o_{1,0}$	$o_{1,1}$	$o_{1,2}$
$o_{1,0}$	–	3	4
$o_{1,1}$	–	–	6
$o_{1,2}$	–	3	–

Setup matrix s(i,i')

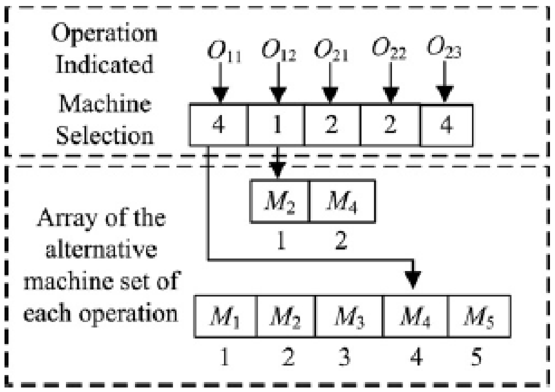
NonDelaySGS1 with $L = \{o_{1,1}, o_{1,2}\}$ or $L = \{o_{1,2}, o_{1,1}\}$
$o_{1,1}$
$o_{1,2}$
0
5
10
15
20
$o_{1,2}$
$o_{1,1}$
$d_1 = d_2$
Optimal solution

Non-dominance of *NonDelaySGS1*.

In 2011, Guohui Zhang et al. [3] The advantage of GA with respect to other local search algorithms is due to the fact that more strategies could be adopted together to find good individuals to add to the mating pool in a GA framework, both in the initial population phase and in the dynamic generation phase (Pezzella et al., 2007). In this paper, the proposed GA adopts an improved chromosome representation and a novel initialization approach, which can balance the workload of the machines well and converge to suboptimal solution in short time.

Chromosome representation

Better efficiency of GA-based search could be achieved by modifying the chromosome representation and its related operators so as to generate feasible solutions and avoid repair mechanism. Ho et al. (2007) developed extensive review and investigated insightfully on chromosome representation of FJSP. Mesghouni, Hammadi, and Borne (1997) proposed parallel job representation for solving the FJSP. The chromosome is represented by a matrix where each row is an ordered sequence of each job. Each element of the row contains two terms, the first one is the machine processing the operation, and the second one is the starting time of this operation.


Operation Indicated
O_{11} O_{12} O_{21} O_{22} O_{23}
Machine Selection
4 1 2 2 4
Array of the alternative machine set of each operation
M_2 M_4
1 2
M_1 M_2 M_3 M_4 M_5
1 2 3 4 5

Machine Selection part.

Population initialization is a crucial task in evolutionary algorithms because it can affect the convergence speed and the quality of the final solution (Shahryar, Hamid, & Magdy, 2007). In this section, we mainly present two methods to solve the first sub-problem through assigning each operation to the suitable machine. These methods take into account both the processing time and the workload of the machines.

Global Selection (GS)

We define that a stage is the process of selecting a suitable machine for an operation. Thus this method records the sum of the processing time of each machine in the whole processing stage. Then the machine which has the minimum processing time in every stage is selected. In particular, the first job and next job are randomly selected. Detailed steps are as follows:

Step 1: Create a new array to record all machines' processing time, initialize each element to 0;
Step 2: Select a job randomly and insure one job to be selected only once, then select the first operation of the job;
Step 3: Add the processing time of each machine in the available machines and the corresponding machine's time in the time array together;
Step 4: Compare the added time to find the shortest time, then select the index k of the machine which has the shortest time. If there is the same time among different machines, a machine is selected randomly among them;
Step 5: Set the allele which corresponds to the current operation in the MS part to k;
Step 6: Add the current selected machine's processing time and its corresponding allele in the time array together in order to update the time array;
Step 7: Select the next operation of the current job, and execute Step 3 to Step 6 until all operations of the current job are selected, then go to Step 8;
Step 8: Go to step 2 until all jobs are all selected once.

In 2007, J. Heinonen et al. [4] Manufacturing today is primarily cooked down to all-out efforts into profitability. Factories are moved to low-salary countries in order to ensure that profits are maintained and stockholders kept happy. Decisions like these are met with debates about moral, ethics and responsibilities that companies have to society, since losing an entire manufacturing plant can be devastating to a community.

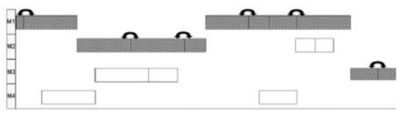
The algorithm consists of two parts. We have the ACO part, where ants crawl over the search space trying to construct a feasible tour. When all ants have constructed their tour, the timestamps are calculated for the individual operations in the schedule defined by a tour, which allows us to calculate the makespan. The postprocessing part springs to life when there is a schedule to operate on. The pheromone update of the ACO occurs only after the postprocessing has finished, this is due to the postprocessing affecting the makespan of the schedule formed by the tour of the ant. After the pheromone update ACO continues with the next iteration.

ACO

ACO belongs to the class metaheuristics. The term metaheuristic is derived from two greek words, heuristic which means ‘‘to find’’ and the prefix meta, which means ‘‘beyond, in the sense of an upper level’’. It has come to mean a high-level strategy for guiding heuristics in a search for feasible solutions as well as a framework that can be specialized to solve optimization problems. ACO is also a successful example of swarm intelligence, whose purpose is to design intelligent multi-agent systems by taking inspirations from the collective behaviour of social insects.
The inspiration for ACO is the behaviour of foraging ants. Ants in nature are capable of finding the shortest path from the nest to a food source without a visual cue. The information concerning food is communicated through an aromatic essence, called pheromone by a process called stigmergy,

which means modification of the environment. A pheromone is any chemical or set of chemicals produced by a living organism that transmits a message to other members of the same species. There are alarm pheromones, food trail pheromones and others that affect behavior or physiology. Pheromone is volatile and evaporates quickly, otherwise nature would be swamped with pheromone scent deposited there during the years. Ant secrete this pheromone while walking and follow, in turn, other pheromone trails laid by other ants, previously passing through that trail. A strong pheromone concentration on a path stimulate the ants to move in that direction. While ants passing through a food source by using a shorter path return to the nest faster than ants taking a longer route, the quantity of pheromone laid down on the shorter path grows faster than on the longer ones, and cause any single ant to bias toward the shorter path. Occasionally there will be the stray ant that takes the longer route, and there can be seen ants that explore other routes to the food and back to the nest as well. The choice of path seems almost probabilistic in nature.

J. Heinonen, F. Pettersson / Applied Mathematics and Computation 187 (2007) 989–998



A sample 4-machine schedule with the critical path marked in grey and possible swap pairs with arrows. The path is made of four blocks with the largest block consisting of four scheduled operations.

In 2008, F. Pezzella et al. [5] GA is a local search algorithm that follows the evolution paradigm. Starting from an initial population, the algorithm applies genetic operators in order to produce offsprings (in the local search terminology, it corresponds to exploring the neighborhood), which are presumably more fit than their ancestors. At each generation (iteration), every new individual (chromosome) corresponds to a solution, i.e., a schedule of the given FJSP instance. The strength of GA with respect to other local search algorithms is due to the fact that in a GA framework more strategies can be adopted together to find individuals to add to the mating pool, both in the initial population phase and in the dynamic generation phase.

Then, a more variable search space can be explored at each algorithm step. The overall structure of our GA can be described as follows:

1. *Coding:* The genes of the chromosomes describe the assignment of operations to the machines, and the order in which they appear in the chromosome describes the sequence of operations. Each chromosome represents a solution for the problem.

2. *Initial population:* The initial chromosomes are obtained by a mix of two assignment procedures (global minimum and random permutation of jobs and machines) and a mix of three dispatching rules (Random, MWR, MOR) for sequencing.

3. *Fitness evaluation:* The makespan is computed for each chromosome in the current generation.

4. *Selection:* At each iteration, the best chromosomes are chosen for reproduction by one among three different methods, i.e., binary tournament, *n*-size tournament and linear ranking.

5. *Offspring generation:* The new generation is obtained by changing the assignment of the operations to the machines (assignment crossover, assignment mutation, intelligent mutation) and by changing the sequencing of operations (POXcrossover and PPS mutation). These rules preserve feasibility of new individuals. Newindividuals are generated until a fixed maximum number of individuals is reached. In our approach, only the new individuals form the mating pool for the next generation, at each algorithm step.

6. *Stop criterion:* Fixed number of generations is reached. If the stop criterion is satisfied, the algorithm ends and the best chromosome, together with the corresponding schedule, is given as output. Otherwise, the algorithm iterates
again steps 3–5.

In 2010, Eugene Levner et al. [6] Cyclic (periodic) scheduling is an effective way to process various manufacturing, computing, and transportation processes, including those where setup and transportation times are significant. Traditionally, periodic scheduling problems in flexible manufacturing systems have been considered separately in two environments, namely the jobshop and the PERT-shop. The cyclic jobshop has two important special cases: the cyclic flowshop and the cyclic robotic shop (these terms will be explained below). In the traditional jobshop environment, setup and transportation times are usually assumed to be insignificant. For instance, modern machining centres can switch tools quickly so the setup times in such a situation may be small or negligible.

In 2011, Liang Gao et al. [7] In this research, the JSP consists of a set of jobs Job = {J1, J2, . . . , Jn} and a set of machines Machine = {M1, M2, . . . , Mm}. The objective is to minimize the makespan, i.e., the completion time of the last job being completed in the system. In the JSP, several constraints and assumptions are made as follows:

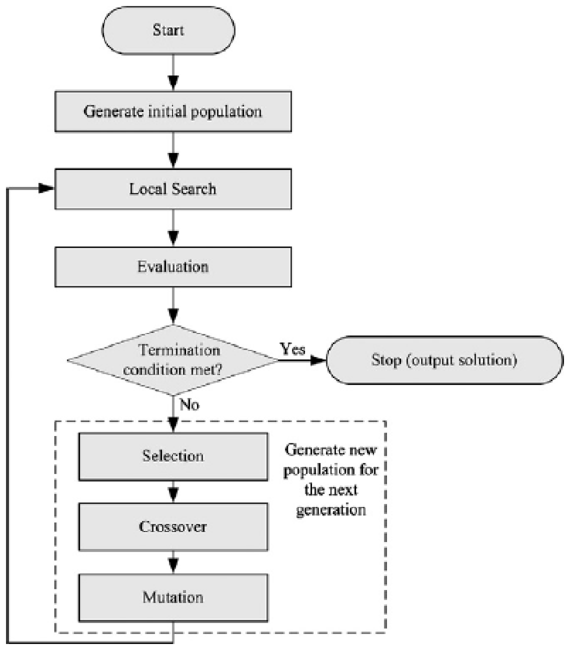
– Each machine could process at most one job at a time.
– Each job is only processed by one machine at a time.
– The sequence of machines which a job visits is completely fixed and has a linear precedence structure.
– All jobs must be processed by each machine only once and there are at most m operations for a job.
– There are no precedence constraints among the operations of different jobs.
– The machines are always available at zero and never break down.
– Processing time of all operations is known.

Job shop scheduling with memetic algorithm

MA could well balance its diversification and intensification to find high quality solutions of the optimization problem.

Diversifi Diversification is a search of different areas of the search space to find the most promising regions. Intensification is a search of the neighborhoods of the individuals to produce better solutions (Ong & Keane, 2004). In the proposed MA, the local search procedure is applied to each child to search for a better solution. The flowchart of the proposed MA in this paper is shown in.

Step 1: Generate initial population. Set parameters of GA including population size, max iteration, mutation probability, crossover probability, etc. Then encode an initial solution into a chromosome. Repeat this step until the number of individual equals to the population size.
Step 2: Apply the local search procedure to improve the quality of each individual.
Step 3: Decode each individual of population to obtain the makespan corresponding with each individual. And compare them to obtain the best solution.
Step 4: Check the termination criteria. If one of the criteria is satisfied, then stop the algorithm and output the best solution; otherwise, go to step 5.
Step 5: Generate new population for the next generation. Genetic evolution with three operators including selection, crossover and mutation is applied to create offspring for the next population. Following this, the algorithm goes back to step 2.


Start
Generate initial population
Local Search
Evaluation
Termination condition met?
Yes
Stop (output solution)
No
Selection
Crossover
Mutation
Generate new population for the next generation

Flowchart of the proposed MA.

In 2011, xiao-yuan wang et al. [8] There are n jobs J_1, J_2, . . . , J_n to be processed successively onmmachinesM_1, M_2, . . . ,M_m in that order. Each job can be processed on no more than one machine at any time, while each machine can handle only one job at a time and the processing of a job may not be interrupted. All the jobs are available for processing at time 0. The operation of job J_j on machine M_i is denoted by T_{ij}. Following Ho et al. (1993), Wang and Xia (2005), and Wang (2007), we assume that the processing time of operation T_{ij} is given by

$P_{ij}(t) = a_{ij}$ (1-bt) (i=1,2..,m; j=1,2..,n).

where $a_{ij} > 0$ denotes the normal processing time of operation T_{ij}, t is its starting time. It is assumed that b satisfies the following condition:

$$0 < b < 1 \quad \text{and } b\left(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} - a_{\min}\right) < 1, \quad \text{where}$$
$$a_{\min} = \min_{i=1,2,\dots,m;j=1,2,\dots,n} \{a_{ij}\}.$$

The first condition ensures that the decrease in processing time of each job is less than one unit for every unit of delay in its starting moment. The second condition ensures that all the job processing times are positive in a feasible schedule (see also Ho et al. (1993), & Wang & Xia (2005) for detailed explanations).

All jobs have the same processing order through the machines and are available for processing at time $t_0 \geq 0$. Let C_{ij} denote the completion time of job J_j on machine M_i in a given permutation. The objective is to minimize the makespan $C_{max} = max_{ij}\{ C_{ij}\}$,i = 1,2,. . . ,m, j = 1,2,. . . ,n, i.e., the maximum of the completion time of all operations. We assume unlimited intermediate storage between successive machines for the general flow shop scheduling problem. We also restrict ourselves to permutation schedules only. Using the threefield notation for scheduling problem classification, the problem can be represented as Fmj | $p_{ij}(t) = a_{ij}$(1- bt)jC_{max}. Let π = ([1], [2] , . . . , [n]) be a permutation of (1, 2, . . . ,n), where *[j]* = I means job J_i is the *j*th one to be processed.

In 2011, leila asadzadeh et al. [9] We proposed an agent-based parallel approach for the job shop scheduling problem. In that model, we developed a multi-agent system containing some agents with special actions that are used to parallelize the genetic algorithm and create its population. We used *JADE* middleware to implement our multi-agent system. Agents distributed over various hosts in network and *JADE* provides a secure communication channel for them to communicate. In this model, each agent has been developed for a special purpose. We can describe them as follow:

• *MA (Management Agent): MA* and *Ai (i=1,2,...,m)* agents have the responsibility of creating the initial population for the genetic algorithm. This agent controls the behaviors of *Ais* and coordinates them in creation step.

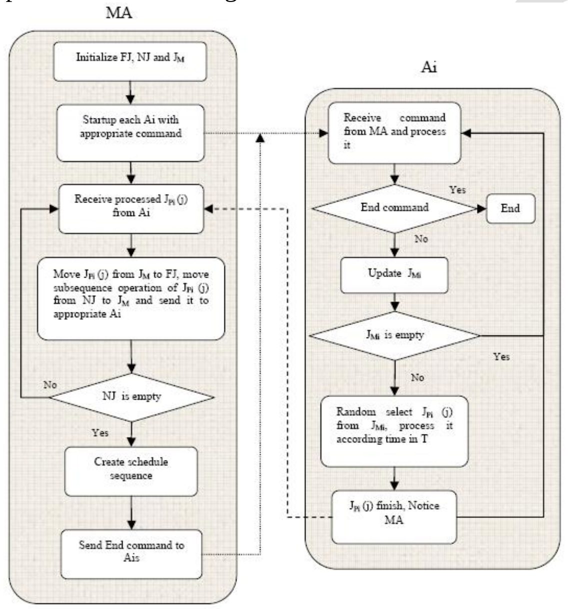
• *Ai (Execute Agent):* Each machine has an *Ai* agent to schedule the operations on it.
• *PA (Processor Agent):* Each *PA* locates on a distinct host and executes genetic algorithm on its sub-population.

• *SA (Synchronization Agent):* This agent locates on main host and coordinates migration between sub-populations of *PA* agents.

In that model, the genetic population is created serially by *MA* and *Ai (i=1,2,...,m)* agents. The sub-populations of *PA* agents are determined and sent to them by *MA*. One disadvantage of this model is the lack of load balancing on the network hosts. On the other hand, the main host that locates the *MA*, *Ai* and *SA* agents is the bottleneck of system and if it crash, the whole multi-agent system will be stopped working.

To solve this problem and improve the performance of creating the initial population, we can extend the model to create sub-populations in a parallel manner. An overall architecture of improved agent-based model has represented in FIGURE. In this model, each host has one *MA* and m *Ai (i=1,2,7,m)* agents. These agents have the responsibility of creating the subpopulation for their host's *PA*.

To synchronize the various processor agents in migration phase, synchronization agent *(SA)* locates on main host and synchronizes them. Parallel creation of sub-populations improves the speed and performance. On the other hand, the division of the population into several sub-populations and sending them to *PAs* can be avoided.


MA
Ai
Initialize FJ, NJ and J_M
Startup each Ai with appropriate command
Receive processed J_Pi (j) from Ai
Move J_Pi (j) from J_M to FJ, move subsequence operation of J_Pi (j) from NJ to J_M and send it to appropriate Ai
NJ is empty
No
Yes
Create schedule sequence
Send End command to Ais
Receive command from MA and process it
End command
Yes
End
No
Update J_Mi
J_Mi is empty
Yes
No
Random select J_Pi (j) from J_Mi, process it according time in T
J_Pi (j) finish, Notice MA

Schedule Process of *MA* and *Ai*

In 2010, Surekha P et al. [10] Ant colonies exhibit very interesting behaviours, though one specific ant has limited capabilities, the behaviour of a whole ant colony is highly structured. They are capable of finding the shortest path from their nest to a food source, without using visual cues but by exploiting pheromone information. While walking, ants can deposit some pheromone on the path. The probability that the ants coming later choose the path is proportional to the amount of pheromone on the path, previously deposited by other ants. This theory was the basis for forming the Ant Colony Optimization (ACO) algorithm using artificial ants. The artificial ants are designed based on the behavior of real ants. They lay pheromone trails on the graph edges and choose their path with respect to probabilities that depend on pheromone trails and these pheromone trails decrease progressively by evaporation.

At the end of each generation, each ant present in the population spawns a complete tour traversing all the nodes based on a probabilistic state transition rule. The nodes are chosen by the ants based on the order in which they appear in the permutation process. The node selection process involves a heuristic factor as well as a pheromone factor used by the ants. The heuristic factor, denoted by $\boldsymbol{\eta}_{\mathbf{ij}}$, and the pheromone factor, denoted by $\boldsymbol{\tau}_{ij}$, are indicators of how good it seems to have node j at node i of the permutation. The heuristic value is generated by some problem dependent heuristics whereas the pheromone factor stems from former ants that have found good solution. The next node is chosen by an ant according to the following rule that has been called pseudo random proportional action choice rule. With probability q_0, where $0\leq q_0<I$ is a parameter of the algorithm, the ant chooses a node from the set of nodes (s) that have not been selected so for which maximizes $(\boldsymbol{\tau}_{ij})^{\alpha}\ (\eta_{ij})^{\beta}$, where $\alpha\geq0$ and $\beta\geq0$ are constants that determine the relative influence of the pheromone values and the heuristic values on the decision of the ant. The probability of choosing the next node is chosen from the set S according to the probability distribution given by:

$$P_{ij}=\frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{h\in S}(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}$$

This probability also known as the transition probability is a trade-off between the pheromone factor and the heuristic factor. The heuristic factor is computed as $\eta_{ij}=1/F(X_j),\, j\in S$, where F(X) represents the cost function of X. While constructing its tour, an ant will modify the amount of pheromone on the passed edges by applying the local updating rule $\tau_{ij}(t)\leftarrow(1-\rho)\tau_{ij}(t)+\rho\tau_0$, where $\boldsymbol{\tau}_{ij}$ (t) is the amount of pheromone on the edge (i, j) at time t; ρ is a parameter governing pheromone decay such that $0<\rho<1$; and τ_0 is the initial value of pheromone on all edges. Once all ants have arrived at their destination, the amount of pheromone on the edge is modified again by applying the global updating rule $\tau_{ij}(t)\leftarrow(1-\rho)\tau_{ij}(t)+\rho\Delta\tau_{ij}(t)$, where $\Delta\tau_{ij}(t)=L^{-1}$ if this is the best tour, otherwise $\Delta\,\boldsymbol{\tau}_{ij}$ (t) =0, and L indicates the length of the globally best tour. The pheromone updating rule was meant to simulate the change in the amount of pheromone due to both the addition of new pheromone deposited by ants on the visited edges and to pheromone evaporation. The algorithm stops iterating either when an ant found a solution or when a maximum number of generations have been performed.

In 2010, Tamer F. Abdelmaguid et al. [11] In GA, the reproduction operator can be seen as an approach for conducting neighborhood search; while, mutation operator provides a mechanism to avoid being trapped in a local optima. The design of both operators is crucial for the success of GA. In the literature, the reproduction and mutation operators applied to the JSP are mainly adopted from the literature of applying GA to the traveling salesman problem (TSP). This adoption is motivated by the similarity between the GA representations used for the JSP and the permutation representation used to encode the sequence of visited cities.

Among the reproduction operators used in the JSP literature are the partial-mapped crossover (PMX), the order crossover (OX) and the uniform or position- based crossover. For both PMX and OX, there are two versions, one in which there are a single crossover point and another one in which there are two crossover points.

The mutation operators used for the JSP implement different mechanisms to exchange the values assigned to randomly selected genes in a given chromosome. Swap mutation, also known as reciprocal exchange mutation, simply exchanges the values assigned to two different randomly selected genes. Inversion mutation, inverts the order of the values assigned to the set of genes located between two randomly selected positions in the chromosome. Insertion or shift mutation selects a gene randomly and sets its value to another randomly selected gene, while the values of the genes between these randomly selected positions are shifted. The displacement mutation is another version of shift mutation in which a substring of genes, instead of a single gene, is moved to a randomly selected new location. Gen and Cheng provide a detailed description of the implementation of the reproduction and mutation operators used in this study.

In 2007, milos seda et al. [12] Flow shop scheduling is one of the most important problems in the area of production management. It can be briefly described as follows: There are a set of *m* machines (processors) and a set of *n* jobs. Each job comprises a set of *m* operations which must be done on different machines. All jobs have the same processing operation order when passing through the machines. There are no precedence constraints among operations of different jobs. Operations cannot be interrupted and each machine can process only one operation at a time. The problem is to find the job sequences on the machines which minimise the makespan, i.e. the maximum of the completion times of all operations. As the objective function, mean flowtime, completion time variance [9] and total tardiness [20] can also be used. The flow shop scheduling problem is NP-complete and thus it is usually solved by approximation or heuristic methods. The use of simulated annealing is presented, e.g., tabu search and genetic algorithms. In a deterministic heuristic is proposed that determines the order of any two jobs in the final schedule based on their order in all two-machine problems embedded in the problem.

Consider three finite sets J, M, O where
J is a set of jobs 1, … , n,
M is a set of machines 1, … , m, and
O is a set of operations 1, … , m.
Denote
J_i … the i-th job in the permutation of jobs
p_{ik} … processing time of the job $J_i \in J$ on machine k.

$(\forall i \in J)$ $(\forall k \in M)$: v_{ik} = waiting time (idle time) on machine k before the start of the job J_i

$(\forall i \in J)$ $(\forall k \in M)$: w_{ik} = waiting time (idle time) of the job J_i after finishing processing on machine k, while waiting for machine k+1 to become free

Define the following decision variables

$$\forall i, j \in J : x_{i\ j} = \begin{cases} 1, & \text{if job } j \text{ is assigned to the } i\text{th position in the permutation, i.e. } J_i = j \\ 0, & \text{otherwise} \end{cases} \qquad (1)$$

The following mathematical formulation of the permutation flow shop scheduling can be derived.

$$\forall i \in J : \sum_{j=1}^{n} x_{ij} = 1 \qquad (2)$$

$$\forall j \in J : \sum_{i=1}^{n} x_{ij} = 1 \qquad (3)$$

$$\forall k \in M - \{m\} : w_{1k} = 0 \qquad (4)$$

$$\forall k \in M - \{1\} : v_{1k} = \sum_{r=1}^{k-1} \sum_{i=1}^{n} p_{i\,r} x_{1\,i} \qquad (5)$$

$(\forall i \in J - \{n\})$ $(\forall k \in M - \{m\})$:

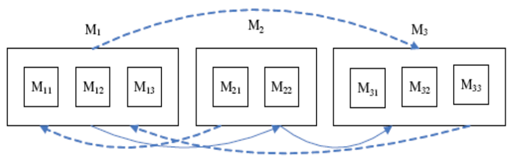
$$v_{i+1,\,k} + \sum_{j=1}^{n} p_{jk} x_{i+1,\,j} + w_{i+1,\,k} = w_{ik} + \sum_{j=1}^{n} p_{j,\,k+1} x_{ij} + v_{i+1,\,k+1} \qquad (6)$$

$$C_{\max} = \sum_{i=1}^{n} (v_{i\,m} + \sum_{j=1}^{n} p_{jm} x_{ij}) \qquad (7)$$

In 2013, Sureshkumar et al. [13] The objective of proposed scheme is to solve a job shop scheduling problem to minimize the makespan time. In order to solve a JSSP artificial intelligence technique genetic algorithm (GA) is used. The genetic algorithm is a probabilistic Meta heuristic technique, which is used to solve optimization problems. They are based on the genetic process of chromosome. Over many generations, natural population evolves according to the principles of natural selection that is survival of the fittest. It starts with the initial solution called population and it is filled with chromosome. Each element in chromosome is called gene. Job is represented by each gene in chromosome and the job sequence in a schedule based on the position of the gene. In our proposed algorithm unordered subsequence exchange crossover (USXX) and shift change Mutation is used.

In 2012, James C. Chen et al .[14] The proposed algorithm is developed in two major modules. Grouping Genetic Algorithm (GGA) is applied to develop machine selection module (MSM) to assign operations to machines. Then Genetic Algorithm (GA) is used to develop operation scheduling module (OSM) to determine the processing sequence of operations on machines. The objectives of the proposed algorithm are the minimization of multiple performance measures including makespan, total tardiness, and total machine idle time. This paper is organized as follows. This section gives an introduction to this research. The next section reviews relevant literature. Section 3 describes the parameters setting for GA and GGA and the detailed procedure of the proposed algorithm. Next, Section 4 presents a case study in a real weapon production factory using the proposed algorithm. Section 5 shows the results and discussion. Finally, Section 6 draws conclusions and gives directions of future work.

J.C. Chen et al./Expert Systems with Applications 39 (2012) 10016–10021



. A typical job shop layout showing the relationship among machine types, parallel machines, and jobs' process routings.

In 2011, DarrellLochtefeld et al. [15] There are several types of EAs including Genetic Algorithms (GAs). GAs model 'survival of the fittest' in order to solve optimization problems. GAs manage a collection of solutions called a population. Individuals represent possible solutions to the optimization problem. Several operators are used on one or more solutions to model survival of the fittest. Individuals are created through the processes of recombination which models the mating of solutions to produce offspring. Offspring compete for a spot in the population through survival selection and are selected to become parents through the process of parent selection. Finally the process of mutation mimics random changes that can occur in nature.

Multi-objective problems have incomparable and often conflicting objective functions. For example, in a factory setting it is often important to both maximize safety and minimize cost. Unless safety levels are translated into a cost by a decision maker, there is rarely a single best solution to the problem. As a result, multiple-objective problems have a multi-dimensional objective space. Even in such a problem space, a rational decision maker would only select a course of action from a subset of possible solutions. A solution may dominate another solution if that solution is better in at least one objective and no worse in all others. Conversely, two solutions can be incomparable in the Pareto sense if each solution has at least one objective value that is better than the other solution. Since a rational decision maker would only pick non-dominated solutions, finding these solutions is the goal of multi-objective optimization. All solutions that are non-dominated by other solutions are also called Pareto efficient solutions.

The process of solving multiple objective problems with EAs is known as Evolutionary Multi-objective Optimization (EMO). MOEAs are the algorithms that perform such optimization. These algorithmsmanage the evolutionary process in a way that produces good and diverse solutions, with a goal of finding diverse solutions on the Pareto efficient frontier. Such methods must balance solution diversity with fitness improvements in two or more objectives.

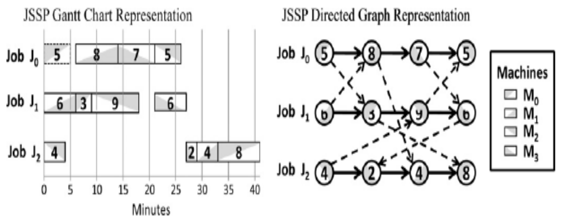
Typically these methods modify the selection of solutions based on Pareto dominance relationships, either through dividing the search hyperspace into hypercubes, or through the direct dominance comparison of solutions in the population. MOEAs are designed to work with multiple objectives making them capable of handling multi-objectivization techniques. The reader is referred to for a complete description of EAs. See also for a more complete description of multi-objective problems, Pareto concepts, and an overview of various types of MOE As.

Multi-objectivization is a term originally coined by Knowles et al. The term describes the process of reformulating a single objective problem into a multiple-objective problem and then solving it with amultiple objective method. There are two types of multi-objectivization methods: those that decompose the original objective into smaller objectives, and those that use new objectives that were not components to the main objective function. Knowles et al. studied hill-climbers in solving a Traveling Salesman Problem (TSP). The original objective was divided into two sub-objectives where the sum of the two sub-objectives equals the original objective. The multiple-objective hill-climbers wereshownto outperform their single-objective hill-climber counterparts. Since the sub-objectives may contain local minima and maxima and those minima and maxima do not always correspond to the global minima and maxima, local minima can be overcome by the multiple-objective hill-climber through picking between solutions in either sub-objective space.

Multi-objectivization is a divide-and-conquer method where a problem is sliced into one or more search spaces. The concept of optimizing a problem through divide-and-conquer techniques is not new. For example, branch-and-bound techniques have been used formally for over half a century in both integer and mixed-integer programming problems. Multi-objectivization by decomposition differs from many past divide-and-conquer methods in that multi-objectivization by decomposition explicitly divides problem objective(s) rather than explicitly dividing the problem search space.

Multi-objectivization has been examined in various areas. Abbass and Deb [10] studied multi-objectivization by adding solution age as an additional objective where solutions were given a birth rank based upon the generation in which they were created.

D.F. Lochtefeld, F.W. Ciarallo / Applied Soft Computing 11 (2011) 4161–4174


JSSP Gantt Chart Representation
JSSP Directed Graph Representation
Job J0
Job J1
Job J2
Minutes
Machines
M0
M1
M2
M3

JSSP schedule representations.

III. CONCLUSION

This paper presents a novel knowledge-based approach for the job shop scheduling problem (JSSP) by utilizing the various constituents of the computational intelligence techniques such as Genetic Algorithm (GA), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). This research focused primarily on discovering new approaches that can match the computational intelligence techniques in solving Job Shop Scheduling problems. Significant improvements can be made by modifying the goals of this paper and adopting techniques to extend the knowledge of job shop scheduling problems. The research dealt specifically with the classical 10x10 job shop scheduling problem with the objective of minimizing the makespan

REFERENCES

[1]. Parviz Fattahi, FariborzJolai, and JamalArkat, “Flexible job shop scheduling with overlapping in operations”, Elsevier Journal of Applied Mathematical Modelling, Vol.33, pp.3076–3087, 2009.

[2]. D.Prot, and O. Bellenguez-Morineau, “Tabu search and lower bound for an industrial complex shop scheduling problem”, Elsevier Journal of Computers & Industrial Engineering,Vol.62,pp.1109–1118,2012.

[3]. Guohui Zhang, Liang Gao, and Yang Shi, “An effective genetic algorithm for the flexible job-shop scheduling problem”, Elsevier Journal of Expert Systems with Applications, Vol.38, pp. 3563–3573, 2011.

[4]. J.Heinonen, and F.Pettersson, “Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem”, Elsevier Journal of Applied Mathematics and Computation, Vol.187, pp.989–998, 2007.

[5]. F.Pezzella, G.Morganti, and G.Ciaschetti, “A genetic algorithm for the Flexible Job-shop Scheduling Problem”, Elsevier Journal of Computers & Operations Research, Vol.35, pp.3202 – 3212, 2008.

[6]. EugeneLevner, VladimirKats, DavidAlcaide López de Pablo, and T.C.E. Cheng “Complexity of cyclic scheduling problems: A state-of-the-art survey”, Elsevier Journal of Computers & Industrial Engineering, Vol.59, pp.352–361, 2010.

[7]. Liang Gao, Guohui Zhang, Liping Zhang, and Xinyu Li, “An efficient mimetic algorithm for solving the job shop scheduling problem”, Elsevier Journal of Computers & Industrial Engineering, Vol.60, pp.699–705, 2011.

[8]. Xiao-Yuan Wang, Ming-Zheng Wang, and Ji-Bo Wang, “Flow shop scheduling to minimize make span with decreasing time-dependent job processing times”, Elsevier Journal of Computers & Industrial Engineering,Vol.60,pp.840–844,2011.

[9]. Leila Asadzadeh and Kamran Zamanifar,"Design and Implementation of a Design and Implementation of a Multi-Agent System for Job Shop Scheduling Problem”, Journal of Computer Science and Security, Vol.5, No.2, pp.287-297, 2011.

[10]. Surekha and Sumathi, "Solving Fuzzy based Job Shop Scheduling Problems using GA and ACO", Journal of Emerging Trends in Computing and information Sciences, Vol.1, No.2, pp.95-102, 2010.

[11]. Tamer F. Abdelmaguid,"Representations in Genetic Algorithm for the Job Shop Scheduling Problem: A Computational Study”, Journal of Software Engineering & Applications, Vol.3, pp.1155-1162, 2010.

[12]. Miloš Šeda, "Mathematical Models of Flow Shop and Job Shop Scheduling Problems", Journal of Mathematical, Computational, Physical and Quantum Engineering, Vol.1, No.7, pp.295-300, 2007

[13]. S.Sureshkumar, G.Saravanan, and S.Thiruvenkadam “Optimizing Make span in JSSP Using Unordered Subsequence Exchange Crossover in GA”, Journal of Computer Engineering, Vol.8, No.5, pp.41-46, 2013.

[14]. James C. Chen, Cheng-Chun Wu, Chia-Wen Chen, and Kou-Huang Chen, “Flexible job shop scheduling with parallel machines using Genetic Algorithm and Grouping Genetic Algorithm”, Elsevier Journal of Expert Systems with Applications,Vol.39,pp.10016–10021,2012.

[15]. Darrell F. Lochtefelda, and Frank W. Ciarallo, “Helper-objective optimization strategies for the Job-Shop Scheduling Problem”, Elsevier Journal of Applied Soft Computing, Vol.11, pp.4161–4174, 2011.