From the Classical Job Shop to a Real Problem: A Genetic Algorithm Approach

Carlos A. Brizuela and Nobuo Sannomiya
Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606–8585 Japan
{cbrizuel,sanmiya}@si.dj.kit.ac.jp
+81–75–724–7467

Abstract

This paper has two main goals. One is to point out the gap existing between the classical job shop problem, for which many procedures have been developed, and a real manufacturing problem that generalizes the job shop, highlighting the points needed to be strengthened in order to get more pragmatic results. The second goal is to design an efficient (acceptable solution quality and fast) method to solve a real problem coming from a manufacturing process. The first goal is achieved by a rigorous definition of both problems emphasizing the differences. The second goal is achieved by applying problem-specific knowledge to the schedule construction method. Numerical experiments are presented as a justification of our second achievement.

1 Introduction

Requirements of small lots of a great variety of products make the planning of a good schedule one of the most important and difficult task in a production process.

The Job Shop Scheduling Problem (JSSP) has been one of the most studied scheduling problem. The Operations Research community has devoted decades of effort trying to find efficient methods for this problem. Recently, many meta-heuristic methods like Simulated Annealing (SA) [13], Tabu Search (TS) [8], Genetic Algorithms (GA) [3], and Neural Networks (NN) [9] have been tested against this complex combinatorial optimization problem. Even though there are some methods to efficiently tackle instances of moderate size, large size instances seem to be very difficult to deal with. Furthermore, this problem is a far-reduced approximation to real manufacturing problems where more complex constraints should be taken into account.

Consider a production system where different types of products should be manufactured. Each product has a certain number of parts that must be processed in a predefined sequence through a set of given machines. This brief consideration of a production system has already, in an informal way, described a problem which includes the classical job shop [13].

Most of the real manufacturing problems are not only complex but also large scale. Due to the complexity of real large-scale scheduling problems it is necessary to have efficient methods for generating, at least, approximated solutions.

In order to have efficient procedures, it is important to have a methodology that takes into account the structure of the problem. The most successful method among the meta-heuristic above mentioned seems to be the TS [8]. The key to its success is the use of a neighborhood construction method which exploits the structure of the problem. For the classical JSSP this is perhaps the right thing to do. However, adding a single constraint to the problem changes things drastically, and then the neighborhood method is no longer of much help. For more constrained cases new methods need to be developed.

We deal with a real manufacturing scheduling problem of which a basic component is the classical job shop problem. A decoding procedure for a GA (in which knowledge of the problem is used) is proposed as a fast solution generation method for the problem. The effectiveness of the proposed algorithms is verified through computer experiments on a real problem data.

The remainder of the paper is organized as follows. Section 2 introduces the problems we are dealing with. Section 3 highlights the GA's literature for these problems. Section 4 presents the method we propose to tackle the problem. Section 5 shows some experimental results, and section 6 states the conclusions of the work.

2 Statement of the problem

We consider a real problem operating in the job shop mode. For describing the problem we start with the

classical job shop and extend it to the problem we are trying to solve.

In the classical JSSP we are given a set of jobs J and a set of machines M. Each job $j \in J$ consists of a set of operations $O_{j\mu_{jk}}$ with $j \in J$, $k \in K_{n_j} := \{1, 2, \cdots, n_j\}$ where n_j is the total number of operations for job j, and μ_{jk} is the machine number to process the k-th operation of job j, i.e. $\mu_{jk} \in \{m_1, m_2, \cdots, m_{|M|}\}$. All machines are different and their processing speeds are constant. The following requirements must also hold:

i. Each job is scheduled on each machine only once.

ii. Each job visits every machine in M.

iii. Each machine can process only one operation at a time.

iv. No machine can free an operation until it is finished (no preemption allowed).

v. The total number of machines of each type is fixed and equals to one.

From the above requirements, we have $n_j = |M|$ for all j in the classical JSSP.

Let us denote the starting time of operation $O_{j\mu_{jk}}$ as $s_{j\mu_{jk}}$, its processing time as $t_{j\mu_{jk}}$. With this notation the problem can be stated as

$$\min_{\sigma \in \Sigma} \max_{j \in J} \{s_{j\mu_{jn_j}} + t_{j\mu_{jn_j}}\} \tag{1}$$

s.t.

$$s_{j\mu_{jk}} > 0 \quad \forall k \in K_{n_j}, j \in J \tag{2}$$

$$s_{j\mu_{jk}} + t_{j\mu_{jk}} \leq s_{j\mu_{jk+1}} \forall k \in K_{n_j - 1}, j \in J \tag{3}$$

$$s_{ja} + t_{ja} \leq s_{ia} \quad \text{or} \quad s_{ia} + t_{ia} \leq s_{ja} \forall i, j \in J; a \in M \tag{4}$$

Here, σ is a schedule in the set of all feasible schedules Σ. This problem belongs to the NP-hard class of problems [4]. Even more, the existence of an approximation algorithm with worst case performance guarantee of $5/4$ of the optimum implies P=NP [14]. It is worth to mention that (1) is not the only optimality criterion used (see [2]).

To state the real problem we want to dealt with, we need to add some constraints to the above representation and modify others in the following way:

i. Each job may be scheduled on each machine more than once.

ii. The jobs do not have to visit every machine in M.

iii. Each machine can process only one operation at a time.

iv. No machine can free an operation until it is finished (no preemption allowed).

v. The number of machines of each type (m_i) is fixed and equals to nm_i (identical parallel machines).

vi. There are special machines (multi-function machines) that can perform activities of a subset of parallel machines. There are ns_i machines of each type sm_i. The different types may have different processing speeds.

vii. The jobs are divided into groups belonging to the same item called the product. There are n different types of products and qp_i products in each type i.

viii. Each product is composed of two kinds of jobs: q_i part-processing jobs, and one assembly job. Every job of any kind has a specified number of operations. There are n_{ij} operations for processing part j (job j) of product i, and n_{i0} operations for the assembly job of product i (see figure 1). No assembly job can start before the processing of all its parts are finished (precedence constraints among jobs).

ix. Each product has a readiness time r_i (the time at which its parts become available), and a due date d_i (the time by which the product should be completed).

x. There is a time window (tw) for processing of jobs (factory working hours). Some machines may have properties such that once it starts inside this time window it can continue until the operation is completed (even though it is finished outside the time window). The other machines must stop processing (when the time window ends) and must resume when the next time window starts.

We consider a case where the number of products of each type is one ($qp_i = 1 \ \forall i \in \{1, ..., n\}$), and the time window is set to 24 hours.

I is the set of types of products with $|I| = n$. The set of all jobs corresponding to all products is $J = J_1 \cup J_2 \cup \cdots \cup J_n$ where J_i is the set of jobs of product i. Their sizes are given by $|J_i| = q_i + 1$. We associate the assembly job of each product with the number zero ($j = 0$) in $\{0, 1, \cdots, q_i\} \in J_i$. The set of machines M is composed of two subsets, the set of parallel machines

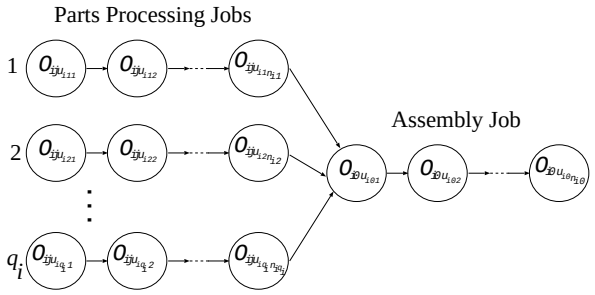

Figure 1: Processing sequence for product i.

PM, and the set of multi-function machines MM with $M = PM \cup MM$.

Now, let us define the starting time of the k-th operation of job j (product i) in machine μ_{ijk} as $s_{ij\mu_{ijk}}$, the corresponding processing time as $t_{ij\mu_{ijk}}$, and the completion time of product i as c_i. Then the problem can be described as

$$\min_{\sigma \in \Sigma} \sum_{i=1}^{n} \max\{0, c_i - d_i\} \tag{5}$$

s.t.

$$s_{ij\mu_{ijk}} > 0 \quad \forall k \in K_{n_{ij}}, j \in J_i, i \in I \tag{6}$$

$$s_{ij\mu_{ij1}} > r_i \quad \forall j \in J_i, i \in I \tag{7}$$

$$s_{ij\mu_{ijk}} + t_{ij\mu_{ijk}} \leq s_{ij\mu_{ijk+1}} \forall k \in K_{n_{ij}-1}, j \in J_i, i \in I \tag{8}$$

$$\max_{j \in J_i - \{0\}} \{s_{ij\mu_{ijn_{ij}}} + t_{ij\mu_{ijn_{ij}}}\} \leq s_{i0\mu_{i01}} \forall i \in I \tag{9}$$

$$s_{ija} + t_{ija} \leq s_{hka} \quad \text{or}$$
$$s_{hka} + t_{hka} \leq s_{ija} \forall j, k \in J; i, h \in I; a \in M \tag{10}$$

If d_i is sufficiently large then (5) loses its meaning and the objective to consider becomes similar to (1). Under this assumption and if there are neither assembly jobs nor multi-function machines and there is only one parallel machine of each type, then the problem becomes the classical JSSP. On the other hand, regardless of what happens in (5) and keeping the assumptions just mentioned, the problem is reduced to a JSSP with due dates (JSSPD). This problem has recently been addressed in a rigorous manner in [1].

If only one product is considered and with no assembly jobs, under the same availability of resources (machines) the problem is called the Flexible Job Shop (FJS). Recently, a few interesting neighborhood construction methods to face this problem appeared in [6].

Since the classical JSSP is NP-hard and it is a special case of the problem defined in (5) - (10), then the latter is also NP-hard. We need to emphasize that besides the classical job shop being a rough approximation to our real problem it does not contribute much in solving the latter. This is because the neighborhood construction method for solving the classical JSSP loses its meaning when the optimality criterion is not the makespan (1), and when the availability of resources is different.

3 GA Approach

Many types of GA's have been proposed for the classical JSSP (see [3], [5], [12], [15], [16]). Their methodologies differ mainly in the problem representation approach or in the way they perform the genetic operations. Among the most successful ones (regarding accuracy) we can cite [15] and [7]. In the first case, the encoding uses knowledge of the problem in order to build neighborhoods to be used in the genetic operations. In the second case dispatching rules and local search are used. Few attempts have been made in trying to solve the generalization given by (5) to (10). Some recent results for the latter problem have been presented in [10] and [11].

The idea presented in these works is basically an extension of previous GA's results for the classical JSSP. The extension is in the proposed individual representation [11] and in a time decomposition of the problem [10]. They obtain good accuracy results after a heavy computational task.

4 The proposed Method

It is well known that most of the computational load in GA's is expended on evaluating the objective function for each individual (solution) of a population. Also when the problem has many constraints the computational effort for performing genetic operations is bigger than in standard situations (because feasibility of solution must be preserved). Therefore, modifying any of these points may help to shorten the computing time.

Our approach to reduce the computational task is based on a very simple representation that requires only simple movements for the genetic operations, and on a decoding procedure that helps to accelerate the convergence of the algorithm. The price we paid is in solution quality when comparing with more time consuming methods.

4.1 Individual Representation

The idea we propose consists of considering the individual to be the ordering of part-processing operations for each product, i. e. a product based representation. Thus, an individual is described as n sequences for the respective products. This consideration is done in order to define simple genetic operations (for faster computation) and to give the decoding part a heavy weight in the schedule construction. Each product is expressed by integers' permutation with repetition representing all operations that have to be performed in order to obtain the product. The job numbers permutation representing product i looks like:

$$Prod(i) = s_{i1}s_{i2}\cdots s_{ip_i}a_{i0}^1\cdots a_{i0}^{n_{i0}} \quad \text{for each } i \in I \tag{11}$$

Each gene s_{it} represents the job number (the allele) of product i, i.e. $s_{it} \in \{1, 2, \cdots, q_i\}$. The order of its operation is given by the number of times the allele is repeated in the sequence (e.g. if gene s_{ik} equals j and it is the h-th time it appears in the sequence, then the gene represents the h-th operation of job j in product i). The a_{i0} represents the assembly job of product i which is repeated n_{i0} times. The total number of part-processing operations p_i is given by

$$p_i = \textstyle\sum_{j=1}^{q_i} n_{ij}.$$

The total number of genes in the array equals to the total number of operations, i.e.

$$q = \textstyle\sum_{i=1}^{n}\sum_{j=0}^{q_i} n_{ij}.$$

We left open the possibility of using a permutation of product numbers $(i_1, i_2, \cdots, i_n \quad \forall i_i \in \{1, 2, \cdots, n\})$ and use the decoding technique for selecting the product's operation to be scheduled.

4.2 Decoding

The decoding procedure we propose here plays a primary role in the efficiency of the algorithm. In order to explain the procedure we need first to state some definitions.

We define in the following description that the indices k and k_{ij} are not time counters but event counters, i.e. k and one k_{ij} are updated every time an operation is scheduled. The time counter is defined as the variable indexed by k and k_{ij}.

Head of a product. The head is defined in the following way. Let $c_{ij}(k_{ij})$ be the completion time of job j (product i) in its k_{ij}-th operation. Then,

$$head_i(k) = \max_{j \in J_i}\{c_{ij}(k_{ij})\} \tag{12}$$

where

$c_{ij}(0) = 0$

$c_{ij}(k_{ij}+1) = c_{ij}(k_{ij}) + t_{ijk_{ij}} \forall i \in I, j \in J_i.$

Every time an operation is scheduled its corresponding index (k_{ij}) is updated. When this happens the index k is also updated. The value in (12) is computed at each iteration of the objective function computation. Thus, we get it for free (i.e. no extra computing time is necessary).

Tail of a product. The tail for a product i is given as follows

$$tail_i(k) = \max_{j \in J_i - \{0\}}\{\alpha_{ij}(k_{ij})\} \tag{13}$$

where

$$\alpha_{ij}(0) = \sum_{k=1}^{n_{ij}} t_{ij\mu_{ijk}} + \sum_{k=1}^{n_{i0}} t_{i0\mu_{i0k}} \forall j \in J_i - \{0\}, i \in I,$$

$$\alpha_{ij}(k_{ij}+1) = \begin{cases} \alpha_{ij}(k_{ij}) - t_{ij\mu_{ij(k_{ij}+1)}} \\ \text{for } 0 \le k_{ij} < n_{ij} \\ \alpha_{ij}(k_{ij}) - t_{i0\mu_{i0(k_{ij}-n_{ij}+1)}} \\ \text{for } n_{ij} \le k_{ij} \le n_{ij} + n_{i0} - 2 \end{cases}$$

For computing the tail, we consider only parallel machines. This is because consideration of multi-function machines will make computation difficult due to the different processing speeds of multi-function machines. In this way at each step k we just need to do a simple arithmetic operation for each job, which is not a time consuming calculation.

The following function gives an idea of the time availability each product has before its tardiness becomes greater than zero.

$$tallow_i(k) = d_i - (head_i(k) + tail_i(k)) \qquad (14)$$

The strategy for choosing a product is based on assigning a probability inversely proportional to the product's time availability. If the product i has already been completed then the assigned probability is zero. On the contrary, if the product is not yet completed we continue as follows. Let $m0$ be a positive constant used for scaling and define a_i as:

$$a_i(k) = \begin{cases} \frac{m0}{|tallow_i(k)|} & \text{for } tallow_i(k) \neq 0 \\ m0/100 & \text{for } tallow_i(k) = 0 \end{cases}$$

then the assigned probability for choosing product i at step k is given by

$$pr_i(k) = \frac{a_i(k)}{\sum_{i=1}^{n} a_i(k)} \qquad (15)$$

This is done in order to generate schedules that tend to minimize the tardiness of each product, and at the same time to avoid repeating the product selection sequence for different individuals. This procedure has some similarity with the Earliest Due Date heuristic (EDD) and can be considered as its stochastic version. Because, instead of deterministically choosing the latest job we choose it probabilistically. In this way we may denote our product selection method (in the decoding procedure) as SEDD (S for Stochastic).

Once a product is chosen the operation to be scheduled is given by the first element in the string sequence not yet scheduled. The way to choose a machine from M is to determine which machine will make the operation start as soon as possible (earliest start time). If, in more than one machine the operation can start at the earliest possible time then the machine with the smaller index is chosen.

Since the individuals are generated through a similar stochastic procedure, differentiating only in the jobs scheduling sequence, the expected diversity should be low, forcing to a fast convergence of the algorithm to suboptimal solutions.

4.3 Reproduction Scheme

The 2/4 selection strategy is used, i.e. two individuals are randomly selected for mating and from the four individuals (two parents and two children) the best two are selected to form a part of the next generation.

For the crossover operator, simple interchange of the two selected individuals' genes, corresponding to the same products, is used (one point crossover). This is better illustrated in figure 2. Here, the genes corresponding to products 1 to k of both individuals are exchanged. Genes corresponding to products $k+1$ to n remain unchanged.

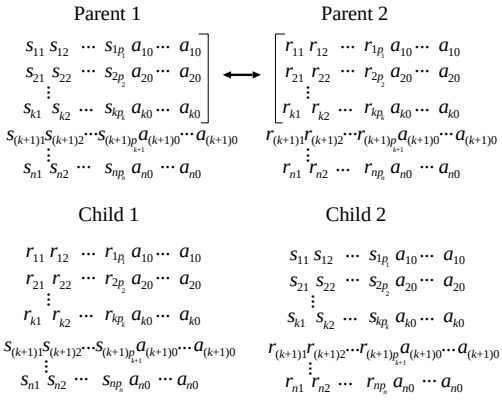

Figure 2: Crossover for the Product based Representation.

The mutation operator is based on a simple swap, and on a segment swap which are explained as follows.

First a product i, $Prod(i) = s_{i1}s_{i2}\cdots s_{ip_i}a^1_{i0}\cdots a^{n_{i0}}_{i0}$ is selected randomly. For the simple swap mutation two numbers between 1 and p_i are randomly chosen. These are indices (loci) of genes to be interchanged. For the segment swap mutation, two segments of genes are randomly chosen and their positions are interchanged.

The crossover rate is given by pc and the mutation rate by $pm = 1 - pc$.

4.4 The proposed Algorithm

The population size is set to be g as a constant value. The overall algorithm can be briefly described as follows.

Algorithm 1. Overall Procedure.

Step 1. Set $r = 0$. Generate an initial population $POP[r]$ of g individuals.

Step 2. Using Random Selection choose two individuals from $POP[r]$ for genetic operations.

Step 3. Perform crossover with probability pc and mutation with probability $1 - pc$.

Step 4. Compute the total tardiness for the parents and the children using the decoding procedure explained in section 4.2.

Step 5. Select the best two individuals (minimum tardiness individuals) from the parents-children set.

Step 6. If the total number of selected individuals is smaller than g, then go to Step 2; otherwise go to Step 7.

Step 7. Set $r = r + 1$. Construct the new generation $POP[r]$ of g individuals.

Step 8. If the stop criterion expires then stop, otherwise go to Step 2.

The results of applying this algorithm to a real problem data are discussed in the next section.

5 Experimental Setup and Results

To analyze the performance of our proposed method we apply it to a real problem data which is an instance of the problem described in section 2. The time window is assumed to be 24 hours, there is one product of each type, there are five multi-function machines such as two of type A and three of type B. Type A machines are 1.2 times faster than type B machines (type B machines and the parallel machines have the same processing speed). The list below gives the details of the problem [11].

Different types of Products ($\|I\| = n$)	14
Total number of jobs ($\|J\|$)	134
Total number of Operations (q)	702
Different types of Parallel Machines	17
Total number of Parallel Machines ($\|PM\|$)	50
Different types of Multi-function Machines	2
Tot. number of Multi-function Mach. ($\|MM\|$)	5
Max. number of operations of a single product	265

The experiment consists of the generation of sets of 100 initial solutions (initial population), i.e. g=100. This value of g is used as a good compromise between computing time and accuracy results. The solutions (individuals) are generated randomly, i.e. the genes given by (11) are random integers uniformly distributed in the closed interval [1,n]. The algorithm described in section 4.4 is applied to the set of initial solutions. The crossover and the mutation rate is given as pc=0.9 and pm=0.1, respectively. The stop criterion used (Step 8, Algorithm 1) is given by a maximum number of iterations (generations) r_{max}=1000. Ten different runs are performed for the experiment. The average over 10 runs, and the best and worst values are obtained along with the average computing time.

The earliest due date heuristic result (for the total tardiness (5)) taken from [10] is 14654 units of time. The average, best, and worst values obtained by our SEDD in $POP[0]$ (randomly generated initial population) are, respectively, 11378.35, 10088 and 14851, over 100 generated solutions. This means that our population of initial solutions alone outperforms the result of the EDD heuristic and the initial populations generated in [10] and [11], which are in the order of 35000 units of time.

The best result reported in [11] is 9045 (unfortunately the mean is not reported). The computation time for this case was 1599 seconds (without taking into account the tuning process).

Table 1 shows our results for two different algorithms. The only difference between them is the decoding procedure. ALGO1 chooses product i with probability given by (15) while in ALGO2 product i is chosen randomly. Notice that no exhaustive tuning is performed and the genetic operations are kept as simple as possible. The computation is performed in a Sun Ultra 60 (2360) machine.

Figure 3 shows the total tardiness convergence curves for the best individual of each algorithm. We can see a better accuracy performance of ALGO1 over ALGO2. This allows us to claim that the stochastic selection of the latest operation at each step in the schedule construction is better than a random selection.

From the experimental results we can say that the merit of the decoding technique we propose here is the capability of generating good solutions when compared to a traditional heuristic procedure and to previous results reported for this problem.

6 Conclusions and Future Research

An efficient decoding strategy that uses knowledge of the problem for a GA based method applied to a tough combinatorial optimization problem has been presented. This procedure generates high quality solutions. In fact, the best random initial solutions on the literature available for this problem are obtained here.

Numerical results show that the decoding method helps to generate good solutions in short computing time.

The differences between the classical JSSP and a natural generalization have been highlighted through a detailed description of the restrictions involved in a real problem. This shows that the classical model is still far away from real world models.

Further experiments need to be carried out, especially to compare how different product-selection methods influence the accuracy results and computing time. Also the use of the proposed decoding technique only for the generation of the initial population will be considered.

Algorithms	Total Tardiness			Computation Time
	Mean	Best	Worst	
ALGO1	9801.1	9675	9883	632.73 seconds
ALGO2	9861.8	9775	9931	238.04 seconds

Table 1: Total Tardiness. g=100. pc=0.9. pm=0.1. Maximum Generations (r_{max})=200.

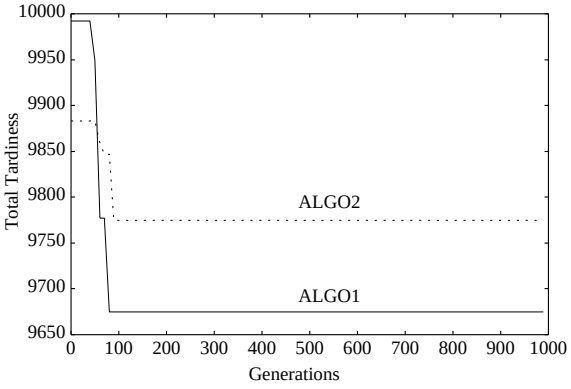

Figure 3: Total Tardiness convergence curves for the best individuals of ALGO1 and ALGO2.

Acknowledgment

This work is partly supported by the Grant-in-Aid for Scientific Researches of the Ministry of Education, Science and Culture of Japan under Grant: B-11450154.

References

[1] E. Balas, G. Lancia, P. Serafini and A. Vazacopoulos. Job Shop Scheduling with Deadlines. *Journal of Combinatorial Optimization*, Vol. 1, pp: 329-353 (1998).

[2] P. Brucker. *Scheduling Algorithms*. Springer (1995).

[3] L. Davis. Job Shop Scheduling with Genetic Algorithms. *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pp: 136-140 (1985).

[4] M. R. Garey, D. S. Johnson and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, Vol. 1, No. 2, pp: 117-129 (1976).

[5] S. Kobayashi, I. Ono, and M. Yamamura. An Efficient Genetic Algorithm for Job Shop Scheduling Problems. *Proceedings of 6th International Conference of Genetic Algorithms*, pp: 506-511 (1995).

[6] M. Mastrolilli and L. M. Gambardella. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, Vol. 3, pp: 3-20 (2000).

[7] D. C. Mattfeld. *Evolutionary Search and the Job Shop*. Physica-Verlag, Heidelberg (1996).

[8] E. Nowicki and C. Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, Vol. 42, No. 6, pp: 797-813 (1996).

[9] I. Sabuncuoglu and B. Gurgun. A Neural Network Model for Scheduling Problems. *European Journal of Operational Research*, Vol. 93, No. 2, pp: 288-299 (1996).

[10] N. Sannomiya, H. Iima, K. Ashizawa and Y. Kobayashi. Application of Genetic Algorithm to a Large-Scale Scheduling Problem for a Metal Mold Assembly Process. *Proceedings of 38th IEEE Conference on Decision and Control*, pp: 2288-2293 (1999).

[11] N. Sannomiya, H Iima, K. Suzuki, and Y. Kobayashi. Genetic Algorithm Approach to a Scheduling Problem for a Complex Manufacturing System. *Proceedings of 8th IFAC Symposium on Large Scale Systems: Theory and Applications*, pp: 271-276 (1998).

[12] G. Shi, H. Iima and N. Sannomiya. A New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm. *Proceedings of 35th IEEE Conference on Decision and Control*, Vol. 4, pp: 4395-4400 (1996).

[13] P. J. M. Van Laarhoven, E. H. L Aarts and J. K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, Vol. 40, No.1, pp: 113-125 (1992).

[14] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevastianov, and D. B. Shmoyds. *Operations Research*, Vol. 45, No. 2, pp: 288-294 (1997).

[15] T. Yamada and R. Nakano. A Fusion of Crossover and Local Search. *IEEE International Conference on Industrial Technology (ICIT '96)*, pp: 426-430 (1996).

[16] T. Yamada and R. Nakano. *Genetic Algorithms in Engineering Systems*, IEE Control Engineering Series 55, pp: 136-140. A. M. S. Zalzala and P. J. Fleming, Editors (1997).