Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

A GA Embedded Dynamic Search Algorithm over a Petri Net Model for an FMS Scheduling

Yung-Feng Chiu *Li-Chen Fu*

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

January 27, 1997

Abstract

In this paper, a genetic algorithm (GA) embedded dynamic search strategy over a Petri net model provides a new scheduling method for a flexible manufacturing system (FMS). The chromosome representation of the search nodes is constructed directly from the Petri net model of an FMS, recording the information about all conflict resolutions, such as resource assignments and orders for resource allocation. The GA operators may enforce some change to the chromosome information in next generation. A Petri net based schedule builder receives a chromosome and an initial marking as input, and then produces a near-optimal schedule.

Due to the NP-Complete nature of the scheduling problem of an FMS, we also propose a dynamic FMS scheduler incorporating the proposed GA embedded search scheme, which generates successive partial schedule, instead of generating a full schedule for all raw parts, as the production evolves.

1 Introduction

In FMS's, generating a schedule to efficiently allocate resources over time for manufacturing of various products is a complex and difficult task. The simulators for FMS's which are tied with heuristic dispatching rules [1, 2, 3] are so expensive with long computing time that they can hardly be taken into consideration for practical applications.

On the other hand, Petri net is recognized to be a very appropriate tool to model discrete event dynamic systems murata. Any method incorporating Petri net to solve the dispatching problems in an FMS starts with Petri net modeling and then employs a heuristic search, like beam search[4] or A^* search[5, 6], to find an optimal or a near-optimal solution over the Petri net model.

Also, research efforts on using GA's for solving job shop problems (JSP's) has a history of a decade [7, 8, 9, 10, 11]. They are, however, all devoted to finding the optimal solutions for the traditional JSP benchmark problems [12]. In a different line of research, approaches have been developed for integrating GA's and simulations to seek efficiently the best combination of dispatching rules in order to obtain an appropriate production scheduling under specific performance measures [2].

In this paper, we propose a genetic search algorithm based on the systematic TPPN model to find an optimal or near-optimal firing sequence from the initial marking to the final marking without the requirement of enormous memory space to store all nodes generated in the search process, like A* search. Most importantly, a GA embedded search based dynamic scheduling strategy is proposed to produce a feasible and near-optimal schedule to resolve the conventional problem with exponential growth of search time vs. the problem size.

The organization of this paper is described as follows. Section 2 presents a Timed Petri net model for an FMS. In Section 3, a GA embedded search method will be employed to solve a well formulated search problem. In Section 4, a dynamic scheduling strategy is introduced to obtain a near optimal schedule. and an example of using the proposed machinery is presented. Finally, conclusions are provided in Section 5.

2 Petri Net Modeling for FMS Scheduling

In our TPPN model, there are two sub-models. One is called Transportation Model which is stationary, and the other is called Process-Flow Model which can be variable [13]. The objective of the Transportation Model is to model the behavior of the AGV traveling from the current stop to its destination stop, and that of the Process-Flow Model is to describe the behavior of part routing and resource assignment. The two sub-models, of course, are interacted with each other to undertake the necessary actions in response to triggering from another.

This research is sponsored by National Science Council, R.O.C., under the grant : NSC 85-2212-E-002-078.

Transportation Model

A layout of the transportation system consists of a set of stops, which represent the locations where the material handling carriers can be stopped and can stay, and a set of path segments, each connecting a pair of adjacent stops. Hence, when the material handling carrier moves, it can only rest at some stop since stopping at any location within a path segment between any pair of stops is prohibited. Here, for simplicity we will consider an FMS where the material handling system is the automated guided vehicle (AGV) system and each machine M_i including the loading station has it own AGV_{m_i}, which has a home position at the machine and is used only to send a part away to another machine. After delivering a part to another machine, the AGV returns to its home position at the owner machine for the next delivery [14, 15]. The basic concept of this sub-model can be found in [13].

Process-Flow Model

A process flow in manufacturing can be seen as a sequence of *part transportation* and *part processing* on NC machines. The Petri net model we proposed to represent such process flow of each part type is called Process-Flow Model, which models the technological precedence constraints for processing parts and normally provides more than one alternative route to accomplish the processing.

Now, an example will be given in the following to illustrate an FMS and as a example to construct its TPPN Model. The layout of this example is shown in Fig. 1. The part processing scenario is described as follows: there are two jobs, namely job J_1 and job J_2, which need to be carried out subjected to the job requirements listed in Table 1. This table provides the technological precedence constraints among the operations. For example, J_1 can be accomplished via two process plans, P_{11} and P_{12}, where the former has three operations, O_{111}, O_{112} and O_{113}, and the latter has two operations, O_{121} and O_{122}. Whereas job J_2 can be carried out only through single process plan consisting of two operations, O_{211} and O_{212}. The operation O_{111} in process plan P_{11} of job J_1 can be run by machine M_1 at the cost of 2 minutes and can also be executed by machine M_2 at the cost of 1 minute, the operation O_{112} can be run by machine M_2 at the cost of 3 minutes or by machine M_3 at the cost of 2 minutes, and the operation O_{113} can be executed by machine M_1 at the cost of 1 minute or by machine M_3 at the cost of 2 minutes. Note that in process plan P_{11} the operation O_{112} can be started only after the operation O_{111} is completed.

3 GA Embedded Search over a Petri Net in the Application for FMS

Genetic algorithm was developed by Holland [16] to study the adaptive process of natural systems and to design artificial system software that imitates the adaptive mechanism of the natural systems. The assumption underlying the use of GA's for scheduling is that optimal solutions will be found in the neighborhood of good solutions.

In general, GA consists of a chromosome representation of the nodes in the search space, a set of simple operations that takes the current population into consideration and generates the successive improved population, a fitness function to evaluate the search nodes, and a set of stochastic assignments to control the genetic operations.

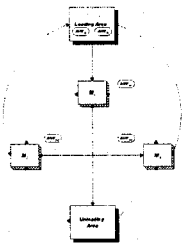

Figure 1: The System Layout of Example

	Job 1		Job 2
Operation	Plan 1	Plan 2	Plan 1
1	$M_1(2)/M_2(1)$	$M_1(5)/M_2(2)/M_1(1)$	$M_1(4)/M_3(2)$
2	$M_2(3)/M_3(2)$	$M_1(3)/M_3(2)$	$M_1(3)/M_2(4)/M_3(4)$
3	$M_1(1)/M_3(2)$		

Table 1: Job Requirements of Example

Notation

Before we explain how to apply the GA embedded search over a Petri net model, we will first introduce the following notation.

- *Conflict place:* A place $p \in P$ with more than one output transitions, i.e., $|p\bullet| > 1$.
- *List:* A special type of list, denoted by l, of which each element is one kind of sequence of the identities of the output transitions associated with a conflict place to express a priority order of those transitions.
- $\|l\|$: The number of elements in the list l.
- c_i: A chromosome, a set of *Lists*.
- C: The set of all possible chromosomes, i.e., the search space of a scheduling problem.
- C_m: The mating pool, tentative new population, for further genetic operation.
- n_{J_i}: The total number of raw parts of a job J_i.
- WIP: Abbreviation of "work in process", the total number of parts remain in the system.
- T_e: The set of all enabled transitions for one marking.

Chromosome Representation

Since a TPPN model of an FMS has encoded each possible processing flow of every part type, i.e., has

modeled the technological precedence constrains for processing a part type, which usually allows more than one alternative route to accomplish the processing, we can extract a conflict place p_i which corresponds to a "plan selection" for a job, say job J_i, from the TPPN model of an FMS and, then, create a list l_{p_i} for the place p_i with $\|l_{p_i}\| = n_{J_i}$. Each element in the list l_{p_i} is randomly assigned to one of the identities of the set $p_{p_i}\bullet$. Every time when a conflict place p_i of this kind contains a token, we fire one, and only one, of its output transitions determined by the code recorded in the list l_{p_i}. Next, we create a list l_{p_j} with $\|l_{p_j}\| = n_{J_j}$ for a conflict place p_j which corresponds to the "resource assignment" for an operation of Job J_j. Each element of the list l_{p_j} is a sequence of the identities of those transitions in the set $p_j\bullet$ to express a priority order of those transitions. Every time when the place p_j has a token, we choose one of its enabled output transitions into the queue q_{R_i}, registering all operations competing for the resource R_i. Last, the representation of a schedule must be able to resolve the conflicts when there are several parts competing for one resource like machine, AGV, and a ticket of a path segment between two adjacent stops, etc. For each resource R_i with $|R_i\bullet| > 1$, we find all operations using this resource R_i, i.e., the set $R_i\bullet$ and, then, create a list l_{R_i} with $\|l_{R_i}\| = WIP$. Each element of the list l_{R_i} is a sequence of the identities of those transitions in the set $R_i\bullet$ to express priority order of these transitions.

A chromosome is generated directly from the Petri net model, and is a set of lists, denoted by $c = \{l_1, l_2, .., l_n\}$. In the following, we summarize the algorithm of constructing a chromosome from a Petri net model.

Algorithm: (Chromosome Representation from a Petri Net Model)

Step 1: Chose a place $p_i \in P$ with $|p\bullet| > 1$,

Step 2: If p_i is corresponding to a plan selection of job J_i,

1. create a list l_i with $\|l_i\| = n_{J_i}$;
2. for each element of list l_i, assign the identity of one transition belonging to the set $p_i\bullet$ randomly to it, and goto *Step 4*.

Step 3: If p_i is corresponding to a resource assignment for an operation of job J_i,

1. create a list l_i with $\|l_i\| = n_{J_i}$;
2. for each element of the list l_i, assign a sequence of all elements in the set $p_i\bullet$ to it randomly.

Step 4: If p_i is corresponding a competition for a resource,

1. create a list l_i with $\|l_i\| = WIP$;
2. for each element of the list l_i, assign a sequence of all elements in the set $p_i\bullet$ to it randomly.

Step 5: Repeat the above steps until every $p_i \in P$ is visited.

Schedule Builder

A schedule builder is dedicated to transforming a chromosome to a feasible schedule and thus, we can utilize it to evaluate the aforementioned indirect chromosome representation. Based on a Petri net model of the system, the evolution of the system can be described by the change of marking in the net. Fig. 2 shows the structure of the schedule builder. The firing sequence of transitions provides the order of the initiation of operations. In the following, we will describe how a schedule builder can use the chromosome to construct a feasible schedule over a Petri net model.

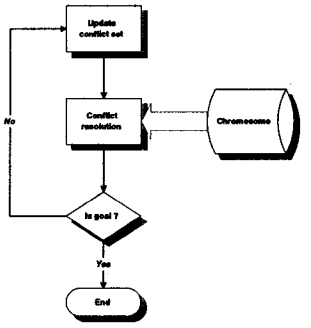

Figure 2: The Structure of Schedule Builder.

Algorithm: (Schedule Builder)

Step 1: Give an initial marking m_0, and a chromosome $c_i \in C$, set the marking $m = m_0$, where m_0 is some preset initial marking, and set the pointer such that it points to the first element of each list l_i.

Step 2: Find the set of all enabled transitions in the marking m, namely, $T_e = \{t_1, t_2, .., t_n\}$.

Step 3: If $T_e = \phi$,

1. find the minimum elapsing time such that one of the busy resource is free;
2. find the maximum elapsing time such that all of the busy resources are free;
3. update the time value between the minimum and maximum elapsing time;
4. goto *step 2*.

Step 4: Get an enabled transition $t_i \in T_e$,

Step 5: If t_i is corresponding to a plan selection for a job,

1. find the corresponding list $l_i \in c_i$;
2. compare t_i with t_j which is currently pointed by the pointer in the l_i;
3. if $t_i = t_j$, fire the transition t_i, and update the pointer of list l_i to the next element;
4. goto *Step 4*.

Step 6: If t_i is corresponding to the resource assignment for an operation,

1. find the corresponding list $l_i \in c_i$;
2. set op_i to t_i according to the element, a priority order set, which is currently pointed by the pointer in the list l_i;
3. goto *Step 4*.

Step 8: Put every op_i to the queue q_{R_i} recording all enabled transitions using the resource R_i.

Step 9: For every queue q_{R_i},

1. find the corresponding list $l_{R_i} \in c_i$;
2. fire transition $t_i \in l_{R_i}$ according to the element, a priority order sequence set, which is currently pointed by the pointer in the list l_i;
3. update the pointers of all lists relating to t_i to the next element.

Step 10: Set $m = m\prime$, where $m\prime$ is the current marking.

Step 11: If the final marking has not been found, then goto *Step 2.*

Fitness Function

The fitness function in genetic algorithms is typically the objective function that we want to optimize in the problem. For the FMS scheduling problem, there exist several performance measures which can be used, such as maximum completion time of all jobs, i.e., makespan, percentage of jobs to meet the due dates, earliness or tardiness, the utilization of machines, or combinations of them. In the current implementation, first, we define the fitness evaluation for each chromosome as follow:

$$f_i = m_{max} - m_i + m_{min},$$

where

m_{max} = the maximum makespan for the current population.

m_{min} = the minimum makespan for the current population.

m_i = the makespan of chromosome c_i in the current population.

Next, we use a linear scaling method to scale the raw fitness value. We calculate the scaled fitness $f\prime$ from the raw fitness using a linear equation of the form:

$$f\prime = a * f + b.$$

In this equation, the coefficients a and b are usually chosen to do two things: enforce equality of the raw and scaled average fitness values and cause maximum scaled fitness to be multi-times (usually twice) of the average fitness. These two conditions ensure that the average population members will receive one offspring copy which is also on average, but the best will receive the designated multiple number of copies [16].

Reproduction

The reproduction operator works on scaled fitness value. This operator, of course, is an artificial version of natural selection, a Darwinian survival of the fittest among chromosome species.

In the implementation, we use the scheme where the probabilities of selection are calculated in the form $pselect_i = f_i/\Sigma f_i$. Then the expected number of chromosome c_i, e_i, is calculated $e_i = pselect_i * population\_size$. Samples of each chromosome are allocated, according to the integer part of the e_i value, into the mating pool. The remainder of the chromosomes needed to fill the population in the mating pool are decided by the fractional parts. The fractional parts of the expected number values are treated as probabilities where, for different chromosomes one by one, the weighted coin tosses are performed by taking the fractional parts as the success probabilities for entering into the mating pool.

Crossover

The chromosome representation described above contains information about plan selections ,resource assignments of every operation, and resource competition resolutions. We can create new chromosomes by exchanging portions of two old chromosomes simply by using the following way.

Algorithm: (Crossover)

Step 1: Randomly select a pair of chromosomes $c_i \in C_m$ and $c_j \in C_m$ and create two new chromosomes $c_i\prime$ and $c_j\prime$.

Step 2: For each list $l_i \in c_i$,

1. get a corresponding list $l_j \in c_j$;
2. toss a weighted coin as the success probability to perform crossover; if it successs, then decides the crossover site cs; else set $cs = \|l_i\|$;
3. assign each element of lists $l_i\prime \in c_i\prime$ and $l_j\prime \in c_j\prime$ to each element of lists l_i and l_j separately and in sequence before the crossover site cs;
4. assign each element of lists $l_i\prime \in c_i\prime$ and $l_j\prime \in c_j\prime$ to each element of lists l_j and l_i separately and in sequence after the crossover site cs.

Step 3: Evaluate the fitness value of the chromosome $c_i\prime$, and the chromosome $c_j\prime$ by the schedule builder.

Step 4: If the fitness value of $c_i\prime$ is not better than that of c_i, replace $c_i\prime$ by c_i.

Step 5: If the fitness value of $c_j\prime$ is not better than that of c_j, replace $c_j\prime$ by c_j.

Step 6: Copy the contents of two old chromosomes c_i and c_j so that $c_i\prime = c_i$ and $c_j\prime = c_j$.

Step 7: Repeat the above steps until the whole chromosomes of successor population are constructed.

Mutation

Mutation can be considered as an occasional (with small probability) random alternation of the structure of a chromosome. One can think of mutation as an escape mechanism for premature convergence. The mutation operator selects a list $l_i \in c_i$ and then changes each element value by tossing the weighted coin. In the implementation, the mutation is embedded into crossover operator, i.e., we modify the algorithm of the crossover operator so that the mutation operator is built into crossover process as follows.

- For each element of each list $l_i \in c_i$,
  1. toss the weighted coin as success probability to do mutation;
  2. if success, reset the element to a valid one randomly.

4 Search Based Dynamic FMS Scheduler

Due to the NP-complete nature of the scheduling problem of an FMS, one must inflict the exponential growth of search time with respect to the problem size. Because the number of parts is high, the use of search techniques in the generation of a schedule for the complete set of parts is a heavy task. So, an approach

called adaptive scheduling [17], has been proposed for high-volume production problems with a new horizon-extension philosophy. The key idea underlying the proposed methodology consists of generating successive partial schedule, a complete schedule for a subset of total raw parts, as the production evolves, instead of generating a full schedule for all raw parts.

Dynamic FMS Scheduler

In our earlier research [18], a limited Work-In-Process (WIP), implementation of the adaptive scheduling algorithm [17], with an A^* search based scheduling method, has been proposed to solve the high-volume FMS scheduling problem. The implementation details are described as follows : First, it defines the WIP for the system. Second, it generates the total schedule segment by segment, of which each segment is a result of running the A^* search, and each A^* search is a full search from the current state (with the number of parts remaining in the system equal to maximum WIP) to a virtual goal state at which the processes of all involved parts are completed.

Now we propose a scheduling policy with the structure consisting of an automatic Petri-net generator, a Petri net model scheduler with GA embedding, a schedule generator, and an extension maker. We define the function of each component and the relations among them.

Automatic Petri net model generator : It generate a timed place Petri net model to describe the behaviors of the part routings and the resource assignments from the file defining the capacity of every machine, possible plans for every job, the relations such as operation vs. machine and operation vs. time, the initial setup information including the starting time, and run time informations of every job, like the number of total raw parts of a job and job priority.

Petri net model scheduler with GA embedding : Based on the Petri-net model generated by the Petri-net generator, it finds a near-optimal schedule in the form of a list of n decisions from the current state to a virtual goal at which the involved parts are completed.

Schedule generator : It generates a partial schedule for a mix of different job types till some of them may meet some predefined condition, and for every job its schedule simply corresponds to the firing sequence over Petri net model, and hence it is dynamic in the sense that the scheduler can re-adjust its previous schedule to adapt to the new system state.

Extension maker : It creates the next search segment by some extension criteria from the current system state.

Implementation

In this subsection, we present the implementation of the proposed dynamic FMS scheduler. First, we define WIP and the condition for the schedule generator, which generates the partial schedule in the current segmented GA search for every job. Before that, three involved parameters are first defined in the following.

Definition 4.1 *Ψ_j stands for the upper bound of the WIP of job J_j in each GA segment search.*

Definition 4.2 *Θ_j is the lower bound of the finished parts of job J_j for a GA segment search.*

Definition 4.3 *Δ_j denotes the number of finished parts of job J_j for a GA segment search.*

For each segmented GA search, Ψ_j is the initial WIP of job J_j. That is to say, before a segment search starts, the number of parts of job J_j in the FMS must be increased to match Ψ_j. The search goal is that every parts in this search are completely processed and are moved out the system. The GA search will then find optimal firing sequences according to the given evaluation function, whereby a part of schedule can be generated. But when the system state evolves to a condition in which $\Delta_j \geq \Theta_j$ for every j, the schedule generation will be halted at that point, and another similar segmented GA search will be initiated again and the number of all the residing parts will all be increased to the respective upper bound, namely Ψ_j.

In the following, we summarizes the procedures of the dynamic scheduling described above.

Algorithm: (Dynamic Scheduling Algorithm)

Step 1: For each uncompleted job J_j, set the number of parts in system of job J_j to match Ψ_j.

Step 2: Create a GA segment search to a get firing sequence.

Step 3: Get a transition t from the firing sequence.

Step 4: Fire the transition t in the TPPN Model.

Step 5: Output a proper schedule command associated with the transition.

Step 6: Repeat *Step 3*, *Step 4* and *Step 5* until every uncompleted job J_j meets the condition $\Delta_j >= \Theta_j$.

Step 7: Repeat the above steps until all jobs have been completed.

Although, the dynamic scheduling strategy may not provide an optimal schedule, when the number of parts to be processed is very high, it can generate an effective schedule in a shorter time interval. In additional, the parameter Ψ and Θ can be used to assign priority to every job. Because Ψs are used to assign number of parts of every job in a segment search, a greater Ψ_j means that the job J_j should have more chances in getting system resources Further, Θs are used to bound the number of finished parts of every job in each segment search, i.e., the minimum number of finished parts of job J_j in a segment search can be no smaller than Θ_j. For example, there are two jobs in the system, J_1 and J_2 with $\Psi_1 = 8$ and $\Psi_2 = 4$, that is to say, for each segment search, there are 8 parts of J_1 and 4 parts of J_2, and hence J_1 has more opportunity to resource allocation than J_2. Now if $\Theta_1 = 3$ and $\Theta_2 = 1$, then the dynamic scheduling algorithm promises that every segment search produce at least 3 parts for J_1 and 1 parts for J_2. As a result from the fact that $\Psi_1 > \Psi_2$ and $\Theta_1 > \Theta_2$, one can generate a schedule in which J_1 appears to have higher priority than J_2. In the extreme condition $\Theta_1 > \Psi_2$ and $\Theta_2 = 0$, one can guarantee that J_1 will always finished before J_2.

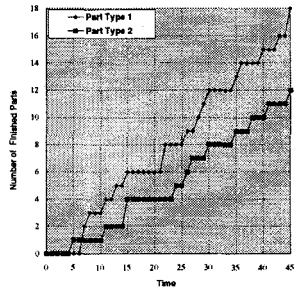

Figure 3: The Relation of Finished Parts and Time for the Example

Example : FMS Scheduling Problem

The layout of the FMS is show in Fig 1 and the job requirements for this example is shown in Table 1. The total number of parts and the priorities of jobs are both shown in the following. In each segmented GA embedded search, the initial WIP of the system is set to be 10.

- The total number of job $J_1 = 18$
- $\psi_1 = 6, \Theta_1 = 4$
- The total number of job $J_2 = 12$
- $\psi_2 = 4, \Theta_2 = 3$

The GA used the following parameters throughout the simulations.

- population size = 30
- crossover probability = 0.7
- mutation probability = 0.05
- maximum number of generation = 100

In Fig 3, the relation between the number of completed parts and the completion time are shown. In average, each segment GA embedded search takes about 3 minutes for one trial on PC 486.

5 Conclusion

In this paper, we consider the FMS scheduling problem. We first use a systematic Petri-net modeling for an FMS, which is composed of two sub-models, Transportation Model and Process-Flow model. Based on this complete Petri net model, a GA embedded search method was applied. The representation of the search nodes (schedules) used in GA is in the form of lists to resolve conflicts for each conflict place in the TPPN model of an FMS. By the genetic search algorithm, a path from the initial marking to the final marking in the reachability graph, generated by the firing rule of the Petri net can be found. This facilitates us to develop a dynamic scheduling method incorporating the GA search. It not only generates an efficient schedule but also allows one to set the different priorities among the jobs. Our experiments show that this approach does represent a good alternative over other techniques for this class of problems.

References

[1] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *Int. J. Production Res.*, vol. 20, no. 1, pp. 27–45, 1982.

[2] H. Fujimoto, K. Yasuda, Y. Tanigawa, and K. Iwahashi, "Applications of Genetic Algorithm and simulation to dispatching rule-based FMS scheduling," in *IEEE Int. Conf. on Robotics and Automation*, pp. 190–195, 1995.

[3] H. Fujimoto, K. Yasuda, and T. Ishimaru, "Evaluation of scheduling rules in FMS using simulator," in *Proceedings of the JAPAN/USA Symposium on Flexible Automation*, vol. 2, pp. 1143–1146, 1992.

[4] H. Shih and T. Sekiguchi, "A timed Petri net and beam search based on-line FMS scheduling system with routing flexibility," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2548–2553, Apr. 1991.

[5] D. Y. Lee and F. DiCesare, "Scheduling flexible manufacturing systems using Petri nets and heuristic search," *IEEE Transaction on Robotics and Automation*, vol. 10, pp. 123–132, Apr. 1994.

[6] C.-J. T., L.-C. F., and Y.-J. H., "Modeling and simulation for flexible manufacturing systems using Petri net," in *Proc. 2nd. Int. Conf. on Automation Technology*, pp. 31–38, 1992.

[7] D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesman: The genetic edge recombination operator," in *Proc. 3rd Int. Conf. on Genetic Algorithms*, pp. 133–140, 1989.

[8] E. Falkenauer and S. Bouffouix, "A Genetic Algorithm for job shop," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 824–829, Apr. 1991.

[9] H.-L. Fang, P. Ross, and D. Corne, "A promising Genetic Algorithm to job-shop scheduling, rescheduling, and open-shop scheduling problems," in *Proc. 5th Int. Conf. on Genetic Algorithms*, pp. 375–382, 1993.

[10] R. Nakano and T. Yamada, "Conventional Genetic Algorithm for job shop problems," in *Proc. 4th Int. Conf. on Genetic Algorithms*, pp. 474–479, 1991.

[11] S. Kobayashi, I. Ono, and M. Yamamura, "An efficient Genetic Algorithm for job shop scheduling problems," in *Proc. 6th Int. Conf. on Genetic Algorithms*, pp. 506–511, 1995.

[12] J. F. Muth and G. L. Thompson, *Industrial Scheduling*. Englewood Cliffs, N. J.: Prentice-Hall, 1963.

[13] T.-H. Sun, C.-W. Cheng, and L.-C. Fu, "A Petri net based approach to modeling and scheduling for an FMS and a case study," *IEEE Trans. on Industrial Electronics*, vol. 41, pp. 593–600, Dec. 1994.

[14] D. Y. Lee and F. DiCesare, "Integrated scheduling of flexible manufacturing systems employing automated guided vehicles," *IEEE Trans. on Industrial Electronics*, vol. 41, pp. 602–610, Dec. 1994.

[15] D. Y. Lee and F. DiCesare, "Integrated models for scheduling flexible manufacturing systems," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 827–832, 1993.

[16] J. Holland, *Adaptation in Natural and Artifical Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[17] C. F. Bispo, J. J. Sentieiro, and R. D. Hibberd, "Adaptive scheduling for high-volume shops," *IEEE Trans. on Robotics and Automation*, vol. 8, pp. 696–706, Dec. 1992.

[18] P. S. Liu and L. C. Fu, "Hierarchical dynamic scheduling for an FMS," in *Proc. 3rd. Int. Conf. on Computer Integrated Manufacturing*, pp. 393–402, May 1992.