Developing Scheduling Policies In Dynamic Job Shops Using Pitts-Based Learning

Muzaffer Kapanoglu
Eskisehir Osmangazi University, College of Engineering, Dept. of Indus. Eng., Eskisehir, 26030 Turkey 90-222-230-3972 ext. 271
muzaffer@ogu.edu.tr

Mete Alikalfa
Eskisehir Osmangazi University, College of Engineering, Dept. of Informatics, Eskisehir, 26030 Turkey 90-222-230-3972 ext. 273
mete@ogu.edu.tr

ABSTRACT
A dispatching policy can be defined as a set of condition-action (CA) rules in which changing job-shop circumstances (such as number of waiting jobs in queues, machine queue lengths in minutes or hours, overall system utilization, machine utilizations, material transport equipment utilizations, work-in-process, and the like) correspond to the condition part and the dispatching rules (shortest processing time first, earliest due date, modified due date, critical ratio, and the like) correspond to the action part. Having considered the fact that advanced manufacturing technologies can enable job shops to practice dispatching policies as efficient as dispatching rules, this paper introduces an intelligent scheduling system that can learn dispatching policies depending on the queue lengths of machines by using Pitts approach of genetics-based machine learning (GBML) for dynamic job shops. In our proposed intelligent scheduling system, the Pitts approach of GBML performs a matching between all possible queue lengths of each machine and dispatching policies (i.e., a set of dispatching rules for each machine) according to the expected efficiencies. In our approach, each dispatching policy (i.e., a set of CA rules) is represented as a chromosome unlike the well-known Michigan approach. The four objectives, all related to minimization of tardiness, are considered: the total tardiness, average tardiness, maximum tardiness, number of tardy jobs. The efficiencies of the dispatching policies are computed by applying the corresponding condition-action (CA) rule-set to the job-shop in a simulation environment. Nine sets of problems (3 settings of tightness of due dates x 3 settings of number of jobs) each with 3 problems were used to compare solutions of our proposed scheduling system with best known nine dispatching rules. The experiments show that dispatching policies learned by the proposed scheduling system outperform the dispatching rules including SPT, EDD, MDD, COVERT, and CR significantly.

Keywords
Intelligent scheduling, dynamic job shop, dispatching policy, genetics-based machine learning

1. INTRODUCTION
The scheduling of manufacturing systems have become more crucial decisions for the companies facing increased competition and/or enhancing their customer satisfaction policies. Nevertheless, most of the manufacturing scheduling problems are known to be NP-hard [1], that is, can not be solved optimally for even modest problem sizes. Aydın and Oztemel [2] states that while most of the methods scheduled the jobs in static manner, the scheduling systems proposed in that manner are not suitable in most real life problems. Thus, dynamic (real-time) scheduling systems appear to be more appropriate against static (off-line) ones.

Due to their ease of implementation and, again, their reduced computational requirement, dispatching rules have been popular approach for a long time. A dispatching rule is a rule that is used to determine which job from the queue is to be operated next. Numerous dispatching rules have been introduced in the literature. Lawrence [3] compares the performance of ten individual dispatching rules with a randomized combination of these rules, and shows that the combined method provides far superior results but requires substantially more computing time. No dispatching rule appears to be superior to all the others [4]. Their efficiency depends on the performance criteria of interest and the operating conditions (e.g., attributes of jobs/machines). As a consequence, the best dispatching rule can not be known a priori in many times. Some researchers adopted simulation to determine the most favourable dispatching rule(s) for the operating conditions, the production objectives, and the current shop status [4]. Pierreval and Mebarki [5] developed a heuristic dispatching strategy, called “shift from standard rules” (SFSR). The strategy proposed is based on a dynamic selection of certain pre-determined dispatching rules. The method for the selection of dispatching rules includes an optimization algorithm for the numerical thresholds obtained via simulation.

Some researchers propose meta-heuristics such as genetic algorithms and more recently, artificial intelligence (AI) methods. Most AI approaches to the scheduling problem have been expert system approaches or heuristic search methods combined with expert knowledge [6, 7] Without an adequate learning mechanism, expert system approaches are perceived as unsuitable for scheduling problems [8]. Over the last two decades, only a limited number of research articles seem to have addressed the use of machine learning approaches for manufacturing scheduling problems among those surveyed. Methods used in these applications include rote-learning, neural networks, induction, and case-based reasoning [9]. Dorndorf and Pesch [10] conducted a probabilistic learning approach, where each gene represents one rule from out of a set of decision rules. The decision rule at the *i*th position means that a conflict in the *i*th iteration should be resolved by using that rule. The algorithm seeks for the best sequence of decision rules for selecting operations to guide the search of a heuristic scheduling algorithm. Hilliard *et al.* [11] first applied classifier systems (Michigan Approach) to discover general rules for job shop scheduling. In their approach, the scheduling rules determine job priorities in a queue. The learning objective is to learn to order jobs in a queue. Lee *et al*. [12] applied genetic algorithms to dispatch jobs at each machine by inducing decision trees, a machine learning technique, to release jobs into shop floor. Aytug *et al*. [13] consider the automated learning of strategies for real-time scheduling in dynamic factory floor environments by using Michigan style GBML. In Aytug's study, learning is used to update the knowledge bases of intelligent dispatchers in the floor shop. Tamaki *et al*. [14] developed a priority equation to set priorities of jobs by using genetics-based machine learning (GBML). Aytug *et al*. [9] provide a comprehensive review of learning methods in manufacturing scheduling.

This paper introduces a method of learning dispatching policies –i.e., a set of adaptive CA rules- depending on queue lengths of machines by using genetics-based machine learning for dynamic job shop scheduling. Pitts approach is adopted where each dispatching policy (CA rule-set) is represented as a chromosome instead of Michigan approach studied by others. Fitness of each chromosome is computed by running a simulation for the corresponding CA rule-set. 9 sets of problems, each with 3 problems, are generated based on different tightness of due dates and different number of jobs to compare solutions of the proposed scheduling system with nine dispatching rules. In the test problems, the objectives used are to minimize the total tardiness, the average tardiness, the maximum tardiness, and the number of tardy jobs.

The remaining of the paper is organized as follows. Section 2 covers definitions and discussions related to job shop scheduling and dispatching rules. Section 3 describes genetics-based machine learning. The framework for the proposed scheduling system using genetics-based machine learning is discussed in Section 4. Experimental results obtained from the proposed scheduling system and nine dispatching rules are compared in Section 5. Section 6 presents the discussion and the interpretation of computational results. The last section includes our conclusions based on the experiments.

2. JOB SHOP SCHEDULING PROBLEM

In a general job shop, there are n jobs Jj (j = 1,.., n) to be processed on m machines Mi (i = 1,.., m) for operations Ojk (k=1,.., nj). Each operation is performed by a predetermined machine for a certain length of time (called processing time) pjk. Jobs may not require all m machines, and may have to visit some machines more than once. A due date dj may be assigned to each job Jj as well.

A job-shop scheduling (JSS) problem is defined as determining the processing sequence of jobs for a given objective such as minimizing makespan, minimizing total tardiness or total number of tardy jobs, maximizing system utilization, etc. The test problems we generated for performance analysis of the proposed scheduling system can be represented as follows according to Pinedo's n/m/A/B notation [15] where n, m, A and B represent number of jobs, number of machines, shop type, and performance measure(s) respectively.:

$$\{50, 100, 200\}/4/G/\left\{T_{sum}, T_{\max}, \overline{T}, \sum_{j=1}^{n} U_j\right\}$$

U_j is 0 if $C_j \le d_j$, and 1 others. The sum of tardiness is formulated below:

$$\text{Tsum} = \sum_j \{\max(0,\ t_j - d_j)\}$$

where tj, and dj mean completion time, and due date, respectively. $T_{\max}$ and $\overline{T}$ are also given as follows respectively:

$$T_{\max} = \max(0,\ t_j - d_j)$$

and

$$\overline{T} = \frac{\sum_j \{\max(0,\ t_j - d_j)\}}{n}.$$

2.1. Dispatching rules

The dispatching rules, also sometimes called as scheduling or priority rules, have been widely used to provide good and time-efficient solutions to job-shop scheduling problems for decades [15, 16].

The dispatching rules can be classified according to their time-dependency and the depth of the information they contain. Time-dependent classification categorizes rules as static and dynamic. Static rules are not time dependent. They are just a function of the job and/or machine data, such as SPT (Shortest Processing Time) rule. The SPT rule assigns higher priorities to jobs with shorter processing times at the current machine. However, dynamic rules are time dependent as it is in MST (Minimum Slack Time) rule. MST is defined as the difference between the due date and the remaining time at the current point. Therefore, the processing priorities of jobs directly depend on their current status.

A second way of classifying rules is according to the information they are based upon, either local or global. A local rule uses only information pertaining to either the queue where the job exists or to the machine where the job is queued [15]. Most of the rules can be defined as local rules, such as FIFO (First In First Out). A global rule uses information regarding other machines, such as the processing time of the job on the next machine to be visited. According to SQNO (Shortest Queue at the Next Operation), a machine selects a job that currently has the shortest queue at the next machine for its following operation.

Nine dispatching rules are considered in this study. These nine rules select the job with minimum Z_{ij} as the next job to be processed on the machine, where Z_{ij} is the priority index, and is defined differently for each rule.

- FIFO (First In First Out): This rule selects the next job to be processed on the machine from the queue based on their arrival time. The formula that calculates the priority of the jobs is $Z_{ij} = r_{ij}$, where r_{ij} is arrival time of job i at machine j.
- EDD (Earliest Due Date): This rule assigns the highest priority to the job with the earliest due date, and ranks

the rest accordingly. The priority formula is $Z_{ij}=d_i$, where d_i is the due date of job i.

- MST (Minimum Slack Time): This rule selects the next job from the queue based on their slack times. Slack time of a job is computed by deducting the current time and the total remaining processing time from the due date of the job. The priority formula is $Z_{ij}=s_i=d_i - rt_i - t$, where rt_i is the remaining process time of job i, and t is the current time.

- SOPN (Minimum Slack Time per Operation): The rule selects the next job from the queue based on their slack time divided by the total remaining operations of the job. The priority formula is $Z_{ij}=(d_i - rt_i - t)/ ro_i$, where ro_i is the total remaining operations of job i.

- SPT (Shortest Processing Time): This rule selects the next job from the queue based on their processing times at the current machine. The priority formula is $Z_{ij}=p_{ij}$, where p_{ij} is the processing time of job i at machine j.

- CR (Critical Ratio): This rule selects the next job from the queue based on their relatively available time divided by the total remaining processing time of the job. The priority formula is $Z_{ij}=(d_i - t)/rt_i$.

- SRPT (Shortest Remaining Processing Time): The rule selects the next job from the queue based on their remaining processing time. The priority formula is $Z_{ij}=r\,t_i$.

- MDD (Modified Due Date): The rule uses a job's original due date as the due date when the job's slack is greater than zero. When the job's slack becomes zero, the earliest finish time act as the modified due date. The priority formula is

$$Z_{ij} = \max\left\{ d_i, t_i + \sum p_{ij} \right\}$$

- COVERT (Cost over Time): COVERT sequences jobs in the descending order of its priority index. W_{iq}, is generally estimated as proportional to its processing time using a parameter b, and it also needs to be adjusted to the worst case by a look-ahead parameter h. If job i queuing for operation j has zero or negative slack then its expected priority is w_i/p_{ij}. If the slack exceeds some worst-case estimates of the remaining waiting time over remaining operations, its expected cost is set to zero. If the slack time is between these two extremes, then the priority goes up linearly as the slack decreases. The priority formula is

$$Z_{ij} = \frac{w_{ij}}{p_{ij}} \times \max\left\{ 0, \frac{h\sum W_{iq} - \max\left\{0, d_i - t - \sum p_{iq}\right\}}{h\sum W_{iq}} \right\}$$

During the last 30 years, the performances of these rules have been studied extensively using varying techniques including simulation. These studies have addressed determining the best rule for different problem settings. At the present time, there is no dispatching rule that is globally better than the others [5, 16] under the varying operating conditions (e.g., tightness of due dates, shop loading rate, etc.) even for a single performance criterion. Any dispatching rule may turn out to be inefficient due to changes in the operating conditions. Therefore, a job-shop management has to address issues of *(i) which dispatching rule can yield better performance, and (ii) what changes in the operating conditions lead to adopting a new rule(s)* by combining dispatching rules and the corresponding shop conditions in which they work efficiently into dispatching policies. In this study, we propose an intelligent scheduling system that resolves the two issues aforementioned by learning dispatching policies including dispatching rules and the corresponding queue lengths. There exist numerous operating conditions that can affect the selection of dispatching rules such as machine queue lengths, system and/or machine utilization, work-in-process, etc [12, 13, 17]. We adopted the queue lengths of machines at a point of time as the key representative of the operating conditions. The proposed scheduling system observes the queue lengths of machines, and learns a common best CA rule set (i.e., the dispatching policy) where each rule is learned to be best for a corresponding queue length. We experimented with four performance criteria, the total tardiness, maximum tardiness, average tardiness, and number of tardy jobs in the test problems.

3. Genetics-based machine learning

Genetics-based machine learning (GBML), developed by Holland, is a machine learning paradigm to automatically discover CA rules for performing desired actions [18]. In contrast to traditional expert systems where rules are handcrafted by knowledge engineers, GBML use genetic algorithms as discovery algorithm to generate and develop CA rules. The CA rules contain two parts; condition part and action part as presented below.

IF <condition> THEN <action>

Whenever the condition part of a CA rule matches the current state of the environment, (i.e., the CA rule is satisfied) then the action part of the CA rule is performed (i.e., the CA rule is fired). That is, CA rules match information about the current state of system and suggest actions, and system moves to a new state. Separating successful CA rules from less successful (or unsuccessful) CA rules, an evaluation (or reward) mechanism needs to be implemented. Therefore, a strength Si(t) is associated with each CA rule i at a time t. The strength is modified by a certain reward mechanism, increasing the strength for successful CA rules, and decreasing the strength for unsuccessful CA rules. The strength measures the influence of a particular CA rule in an evolutionary process, driven by a GA. GA seeks to improve the system's overall performance by replacing unsuccessful CA rules with successful ones. These CA rules are evaluated in an environment, and are automatically assigned strengths based on system's performance. As a result, GBML learns by interacting with an environment from which it receives feedback in the form of numerical reward. Learning is achieved by generating the CA rule set that maximizes the amount of reward received.

GBML is classified into two approaches: the Michigan and Pitt approaches. In the former approach, a CA rule is represented as a chromosome of genetic algorithms (GA), and credits are allocated to CA rules according to their performances obtained by applying a CA rule-set. The CA rule is generated and improved by applying GA at certain time or iteration intervals, where the fitness of each individual is calculated according to its credit. On the other hand, in the Pitt approach, a CA rule-set is represented as a chromosome of GA, and the fitness of an individual is calculated based on experiments, e.g. the simulation, by applying a CA rule-set.

Among the two GBML approaches, Michigan and Pitt, we adopted the latter which represents a CA rule-set as a chromosome, and calculates the fitness of a chromosome based on experiments, e.g. the simulation of the job-shop for a given CA rule-set.

4. PROPOSED SCHEDULING SYSTEM

In our intelligent job-shop scheduling system, each chromosome contains a list of dispatching rules that are associated with different queue lengths. Chromosomes survive according to their performances computed through system simulations. In the proposed system, no machine or workstation has a predetermined dispatching rule, rather a set of CA rules where each CA rule is learnt to be the best for a given queue length and performance criterion.

4.1. Representation of a chromosome

In the proposed approach, a CA rule-set is represented as a chromosome including a set of n dispatching rules, $\boldsymbol{R} = \{R_1,R_2,...,R_n\}$ and a set of k queue lengths $\boldsymbol{Q} = \{Q_1, Q_2,.., Q_k\}$. The designed chromosome contains the dispatching rules. Leftmost rule in the chromosome corresponds to the fact that a single part is waiting in the queue, left second rule corresponds to the fact that two parts are waiting in the queue, and so on. As a generic representation of this structure, the proposed system matches the queue lengths and dispatching rules, each taken from out of their supersets, by employing condition-action pairs in the form of *IF Q_i THEN R_j*. Therefore, chromosomes are designed to support this matching process as illustrated in Fig 1.

MST	FIFO	SOPN		#	SPT

Figure 1. Chromosomal representation of a dispatching policy

The chromosome presented in Fig 1 is interpreted in Fig 2 where n_j is the number of jobs in the queue at any moment.

IF n_j=1 THEN MST

IF n_j=2 THEN FIFO

IF n_j=3 THEN SOPN

:

IF n_j=n-1 THEN #

IF n_j=n THEN SPT

Figure 2. Decoding of the chromosome presented in Fig.2

The last CA rule in Fig 2 is interpreted as follows. If there are n jobs in the queue when the machine is available, the next job to be processed at this machine will be selected according to the SPT rule. The CA rule before the last in Fig 2, on the other hand, means that if there are n-1 jobs on the queue when the machine is available, the next job to be processed at the machine will be selected based on the previously used rule.

4.2. Calculation of fitness

Each individual is evaluated based on the schedules obtained by running a simulation for the corresponding dispatching policy on the test problems. The fitness values are the objective function values computed with respect to equations in Section 2. In this study, the objective is to minimize the total tardiness, average tardiness, maximum tardiness and number of tardy jobs.

4.3. Operators

The operators of the genetic algorithm used to improve our solutions are described as follows.

Selection and reproduction: The tournament selection with the elitist strategy is adopted. In the tournament selection, a small group of individuals (related to tournament size) is sampled from the population, and the individual with the best fitness is chosen for reproduction. Elitism appears to eliminate the destruction of the best solution due to selection. In elitism, the fittest member of a population should always survive and also be present in the next generation. We applied the tournament selection with elitist strategy in this study.

Crossover: Two individuals are matched randomly in a population, and the crossover is performed with respect to crossover rate p_c. Simple, single-point crossover is used in this paper. In single-point crossover, one crossover point is selected. String from the beginning of the chromosome to the crossover point is copied from the first parent; the rest is copied from the other parent for the first child. For the second child, first part of the second parent and second part of the first parent is merged.

Mutation: A gene is selected for mutation with mutation rate p_m. In our proposed approach, the mutation operator changes the CA rules with respect to a uniform distribution when applied.

4.4. The proposed scheduling system

The algorithm of the proposed learning scheduling system is outlined as follows:

I. Generate N number of CA rule-sets randomly, and encode the CA rule-sets as chromosomes. Set iteration counter to 1 (i.e., t = 1), and initialize the upper bound of generation n_g.

II. Apply each corresponding CA rule-set to the problem via simulation. Calculate the fitness value according to the performance criteria based on the simulation results.

III. If $t<n_g$, go to Step IV. Otherwise go to Step V.

IV. Apply genetic operators (selection, crossover, mutation) to the population and generate new population. Set t = t + 1 and go to Step II.

Terminate and select the best CA rule-set.

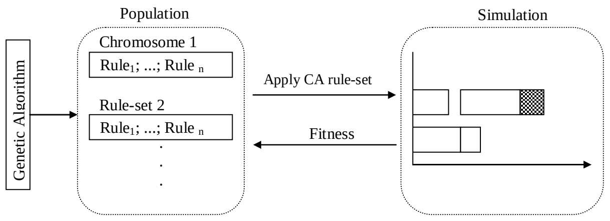


Figure 4. Proposed learning job shop scheduling system

In this study, we propose an intelligent scheduling system that (i) builds dispatching policies considering the queue length as the major attribute of a dynamic job shop, and (ii) minimizes average tardiness, maximum tardiness, total tardiness, and number of tardy jobs by utilizing Pitt approach of genetics-based machine learning.

5. Computational experiments

In the experiments of our study, we determine a dispatching policy to minimize the total tardiness, the average tardiness, the maximum tardiness and the number of tardy jobs in a dynamic job shop. Experiments were conducted on randomly generated problems to test the performance of the intelligent scheduling system. The processing times for each machine are presented in Table 1 [2].

Machine	Distribution
1	Uniform [2, 9]
2	Uniform [2, 9]
3	Uniform [2, 9]
4	Uniform [2, 9]

Table 1. The processing time distributions of machines

Inter-arrival times between jobs are determined according to an exponential distribution using the following rule:

$$t = 20 \cdot \frac{1}{4.5} \cdot \exp(RN)$$

where RN is the random number and *t* represents the inter-arrival time (Aydın and Öztemel 2000).

Due dates are identified with the following rule:

$$\mathrm{DD} = r_i + k \cdot \sum_{i=1}^{n} p_i, \quad k = 2, 4, 6$$

where DD, *r*, *n*, *k* and p_i represent due date, arrival time, the number of operations, coefficient of tightness, and processing time of *i*th operation, respectively (Aydın and Öztemel 2000). 50, 100, and 200 jobs are used in the every test problem.

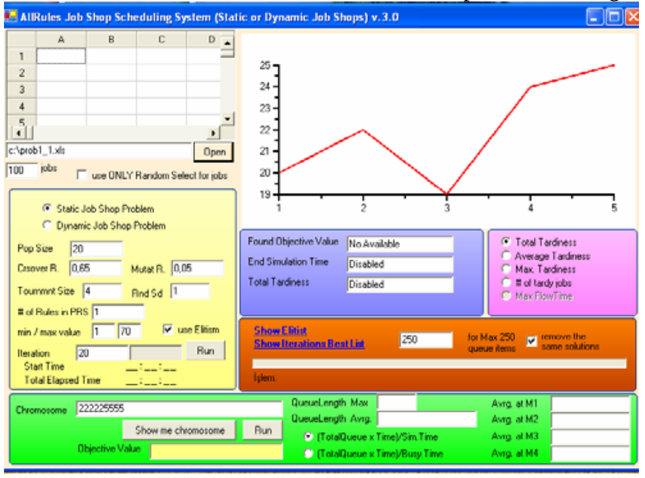
Nine sets of problems (three settings of tightness of due dates x three settings of number of jobs) each with three problems are used to compare solutions of our intelligent scheduling system with best known nine dispatching rules, i.e. FIFO, EDD, MST, SOPN, SPT, SRPT, CR, COVERT, and MDD.

Parameter settings of GBML-based learning system are given in Table 2 as a result of our pilot experiments.

Parameter	*Value*
Population Size	100
Crossover Rate	0.65 (% 65)
Mutation Rate	0.05 (% 0.5)
No. Of Iterations	1000
Tournament Selection Size	4

Table 2. GBML parameter settings

The proposed intelligent scheduling system has been developed in C#. The main screen of the GBML-module is presented in Fig.3.


AllRules Job Shop Scheduling System (Static or Dynamic Job Shops) v.3.0

c:\prob1_1.xls
100 jobs
use ONLY Random Select for jobs

(●) Static Job Shop Problem
( ) Dynamic Job Shop Problem

Pop Size: 20
Crsover R.: 0,65
Mutat R.: 0,05
Tournmnt Size: 4
Rnd Sd: 1
# of Rules in PRS: 1
min / max value: 1 / 70
[✓] use Elitism
Iteration: 20
Start Time: __:__:__
Total Elapsed Time: __:__:__

Found Objective Value: No Available
End Simulation Time: Disabled
Total Tardiness: Disabled

(●) Total Tardiness
( ) Average Tardiness
( ) Max. Tardiness
( ) # of tardy jobs
( ) Max FlowTime

Show Elitist
Show Iterations Best List
250 for Max 250 queue items
[✓] remove the same solutions
Işlem:

Chromosome: 222225555
QueueLength Max
QueueLength Avrg.
(●) (TotalQueue x Time)/Sim.Time
( ) (TotalQueue x Time)/Busy Time
Objective Value
Avrg. at M1
Avrg. at M2
Avrg. at M3
Avrg. at M4

Figure 3 Main screen of the developed GBML system

The dispatching policies obtained from our intelligent scheduling system were compared with the selected dispatching rules in ARENA™ software with VBA™ programming. The experimental results are presented for each group of test problems (k=2, 4, 6) in Table 4a, 4b and 4c in terms of the improvements obtained by using dispatching policies built by our intelligent scheduling system. The improvements are computed based on the average objective values obtained from our generated dispatching policies, and the objective value obtained from the best-performing dispatching rule, i.e., *Improvement (%)* = $100 - \frac{\overline{\mathrm{T}}\left[\text{Dispatching Policy}\right].100}{\overline{\mathrm{T}}\left[\text{Best performing dispatching rule}\right]}$

for the average tardiness as objective.

K=2	Min – Max Improvements (%)							
	T_{sum}		T_{ave}		T_{max}		# of Tardy Jobs	
	Min	Max	Min	Max	Min	Max	Min	Max
50 jobs	6.31	10.58	3.37	9.57	4.54	16.79	6.06	23.80
100 jobs	3.24	11.38	3.24	15.83	0.55	11.69	4.91	9.23
200 jobs	3.08	6.90	3.08	6.90	1.30	5.62	2.22	4.25

Table 4.a Improvements for k=2

k=4	Min – Max Improvements (%)							
	T_{sum}		T_{ave}		T_{max}		# of Tardy Jobs	
	Min	Max	Min	Max	Min	Max	Min	Max
50 jobs	13.46	20.0	13.4	20.0	4.4	27.2	16.6	40.0
100 jobs	4.6	38.0	4.6	38.0	5.5	20.4	8.3	28.0
200 jobs	10.1	16.4	10.1	16.4	0.0	8.0	0.0	3.3

Table 4.b Improvements for k=4

k=6	Min – Max Improvements (%)							
	T_{sum}		T_{ave}		T_{max}		# of Tardy Jobs	
	Min	Max	Min	Max	Min	Max	Min	Max
50 jobs	0.0	100.	0.0	100.	0.0	100.	0.0	100.
100 jobs	5.7	33.7	5.7	33.7	7.3	28.2	30.4	41.1
200 jobs	10.0	16.4	6.7	16.4	3.7	5.6	2.9	15.5

Table 4.c Improvements for k=6

The computational times of our intelligent scheduling system are presented for each group of test problems (k=2, 4, 6) in Table 5.

		Min – Max Time (minute:second)			
		T_{sum}	T_{ave}	T_{max}	# of Tardy Jobs
k=2	50 jobs	1:30 – 1:35	1:43 – 1:52	1:30 – 1:37	1:29 – 1:37
	100 jobs	2:55 – 3:13	3:24 – 3:45	2:59 – 3:17	2:59 – 3:39
	200 jobs	6:07 – 6:20	7:03 – 7:15	6:12 – 6:21	6:17 – 6:39
k=4	50 jobs	1:30 – 1:37	1:43 – 1:51	1:29 – 1:38	1:29 – 1:37
	100 jobs	3:00 – 3:07	3:28 – 3:35	3:00 – 3:06	3:03 – 3:10
	200 jobs	6:01 – 6:09	6:57 – 7:05	6:03 – 6:08	6:05 – 6:09
k=6	50 jobs	1:30 – 1:39	1:48 – 1:55	1:34 – 1:38	1:34 – 1:39
	100 jobs	3:01 – 3:13	3:28 – 3:41	3:00 – 3:11	3:01 – 3:13
	200 jobs	6:01 – 6:09	6:57 – 7:05	6:04 – 6:10	6:02 - 6:09

Table 5 Computational times for each set of problems

6. Interpretation and Conclusion

The computational results of the experiments presented in the previous section are interpreted to determine the performance of the proposed learning scheduling system against dispatching rules in this section. For each set of problems, first we determined the best-performing rule from out of nine dispatching rules. Then we generated a dispatching policy for each case by running our intelligent scheduling system. The performance of the dispatching policies obtained from our intelligent scheduling system for nine problem sets significantly outperforms the best-performing dispatching rule as seen on Table 4 a-c.

In this study, we propose a new intelligent scheduling system for dynamic job shops. The proposed scheduling system adopts (i) a Pitts-GBML as learning mechanism, and (ii) a simulation for evaluations of schedules to construct dispatching policies depending on system attributes. Pitts-based GBML generates a population of dispatching policies each containing of a list of dispatching rules that will be applied when a condition on the system attribute is satisfied. Simulation is performed to collect all performance measurements that will be used in the evaluation of schedules. The dispatching policies determined by the scheduling system contain a dispatching rule for each system attribute which is taken as queue lengths in the experiments. Therefore, as the queue lengths vary, the dispatching rule that will be applied varies according to the corresponding CA rules. The proposed scheduling system has been experimented on 27 test problems for the objective of minimizing tardiness in a job shop with four machines. The results based on the test problems have been very impressive for the dispatching policies constructed by our scheduling system for a dynamic job shop environment by outperforming the selected-best dispatching rules prominently. Building one dispatching policy that applies to all machines can be considered as another success factor in the real shop-floor applications.

The proposed scheduling system incorporates the practical success of dispatching rules into the learning paradigm of GBML. Our experiments demonstrate the strength and robustness of learning scheduling system for dynamic job-shop scheduling problems. The experiments related to the different objective functions than tardiness, impacts of multi-objectivity in different manufacturing configurations including material handling considerations should be addressed in the future researches.

4. REFERENCES

[1] Lageweg, B.J., Lenstra, J.K. and Rinnooy Kan, A.H.G., Job shop scheduling by implicit enumeration, *Management Science*, (1977), **24**, 441-450.

[2] Aydın, M.E. and Öztemel, E., Dynamic job-shop scheduling using reinforcement learning agents, *Robotics and Autonomous Systems*, (2000), **33**, 169-178.

[3] Lawrence S., *Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*, Technical Report, (1984) (Graduate School of Industrial Administration, Carnegie Mellon University).

[4] Kouiss, K., Pierreval, H. and Mebarki, N., Using multi-agent architecture in FMS for dynamic scheduling, *Journal of Intelligent manufacturing*, (1997), **8**, 41-47.

[5] Pierreval, H. and Mebarki, N., Dynamic selection of dispatching rules for manufacturing system scheduling, *International Journal of Production Research*, (1997), **35**, 1575-1591.

[6] Alpar, P. and Srikanth, K.N., A comparison of analytic and knowledge-based approaches to closed-shop scheduling, *Annals of Operations Research*, (1989), **17**, 347-362.

[7] Bel, G., Bensana, E., Dubois, D., Erschier, J. and Esquirol, P., A knowledge-based approach to industrial job-shop scheduling. In *Knowledge-Based Systems in Manufacturing*, edited by A. Kusiak, pp. 207-246, (1989) (Taylor & Francis: London).

[8] Yih, Y., Trace driven knowledge acquisition for rule based real time scheduling systems, *Journal of Intelligent Manufacturing*, (1990), **1**, 217-230.

[9] Aytug, H., Bhattacharyya, S., Koehler, G.J. and Snowdon, J.L., A review of machine learning in scheduling, *IEEE Transactions on Engineering Management*, (1994), **41**, 165-171.

[10] Dorndorf, U. and Pesch, E., Evolution based learning in a job shop scheduling environment, *Computers and Operations Research*, (1995), **22**, 25-40.

[11] Hilliard, M.R., Liepins, G.E. and Palmer, M., Machine learning applications to job shop scheduling, in *Proceedings of the first international conference on Industrial and engineering applications of artificial intelligence and expert systems*, *Tullahoma*, TN, US, (1988), pp. 728-737.

[12] Lee, C.Y., Piramuthu, S. and Tsai, Y.K., Job shop scheduling with a genetic algorithm and machine learning, *International Journal of Production Research*, (1997), **35**, 1171-1191.

[13] Aytug, H., Bhattacharyya, S. and Koehler, G.J., Genetic learning through simulation: An investigation in shop floor scheduling, *Annals of Operations Research*, (1998), **78**, 1-29.

[14] Tamaki, H., Sakakibara, K., Murao, H. and Kitamura, S., Rule acquisition for production scheduling: genetics-based machine learning approach to flexible shop scheduling, in *Proceedings of The Society of Instrument and Control Engineers (SICE) Annual Conference 2003*, Japan, (2003) pp. 252-257.

[15] Pinedo, M., *Scheduling: Theory, Algorithms, and Systems*, (1995) Prentice-Hall: Englewood Cliffs, NJ.

[16] Blackstone, J., Phillips, D. and Hogg, G., A state-of-the-art survey of dispatching rules for manufacturing job shop operations, *International Journal of Production Research*, (1982), **20**, 27-45.

[17] Bhattacharyya, S. and Koehler, G.J., Learning by objectives for adaptive shop-floor scheduling, Decision Sciences, (1998), **29**, 347-376.

[18] Holland, J.H., Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press, (1975)