Adaptive Scheduling in Flexible Manufacturing Systems

Sang Chan Park* · Narayan Raman** · Michael J. Shaw**

ABSTRACT

This paper develops an adaptive scheduling policy for flexible manufacturing systems. The inductive learning methodology used for constructing this state-dependent scheduling policy provides an understanding of the relative importance of the various system parameters in determining the appropriate scheduling rule.

Experimental studies indicated the superiority of the suggested approach over the alternative approach involving the repeated application of a single scheduling rule for randomly generated test problems as well as a real system, and under both stationary and nonstationary conditions. In particular, its relative performance improves further when there are frequent disruptions, and when disruptions are caused by the introduction of tight due date jobs, one of the most common sources of disruptions in most manufacturing systems.

1. Introduction

A practical consideration for many manufacturing systems is their ability to withstand constant disruptions such as the unexpected arrival of "hot" jobs with shorter due dates, machine breakdowns, etc. Consequently, an important measure of the effectiveness of any scheduling approach is its robustness in the face of such disruptions. While there is some recent research dealing with the generation of robust scheduling rules, it has largely addressed static systems only. Previous investigations of scheduling rules in dynamic systems have largely considered their performance of any dispatching rule observed under these conditions will continue to hold when there

* Department of Industrial Management, Korea Advanced Institute of Science and Technology, 373-1 Kusong-dong, Yusong-gu, Taejon, KOREA, 305-701, SANGPARK@cais. kaist. ac. kr, phone: 82-42-869-2920, fax: 82-42-869-2910
** Department of Business Administration, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

are frequent disruptions.

This paper proposes an adaptive approach in which the dispatching rule selected at a given point in time is determined by the existing state of the system. There has been significant research done in identifying dispatching rules that are superior. The major finding is that there is no one rule that clearly dominates others. In order to explain the somewhat inconsistent results obtained by various researchers, Baker (1984) proposes that the relative effectiveness of any rule depends upon the system parameters incorporated in the study. For the objective of minimizing mean job tardiness, he identifies due date tightness and system utilization levels as two parameters that could result in crossovers among the various dispatching rules in a dynamic job shop. His results suggest that it may be possible to improve system performance by implementing a scheduling ***policy*** rather than a single dispatching rule. Such a policy should be adaptive in that it should be able to identify the current ***state*** of the manufacturing system, and should then be able to decide upon the appropriate dispatching rule to be used.

The required mapping rules between the current state of the manufacturing system and the appropriate dispatching rule are constructed through an inductive learning approach. This approach utilizes a set of training examples to construct the knowledge base in the form of a decision tree. There are two major advantages in using inductive learning. First, it can be readily modified to incorporate system-specific attributes both in qualitative and quantitative terms. In so doing, it satisfies a major requirement of any scheduling approach that can be implemented in a real system. Second, it has the capability of incremental learning that enables it to accommodate new attributes as well as new dispatching rules as they become available. It is, therefore, able to constantly improve upon its performance. While, theoretically, these two features can be incorporated in many other approaches, such as regression, the cost of doing so is usually quite high because it entails generating the dependence relationship afresh. In inductive learning, however, the acquisition of incremental knowledge can be incorporated in a modular fashion.

The proposed pattern-directed scheduling (PDS) methodology is general enough to be used in a wide variety of manufacturing scenarios. In this paper, we illustrate the use of this approach for solving the mean tardiness problem in a dynamic flexible manufacturing system (FMS). In this study, first, we augment the dispatching rule selection tree with a meta-tree in order to reduce system nervousness. This meta-tree comprises decision rules that map the current state of the system to a smoothing constant which, in its turn, determines the threshold level required for effecting any change in the dispatching rule used. Second, we propose a systematic approach for incorporating incremental learning towards rule refinement in the decision trees. Third, we extend the investigation of the relative performance of the proposed methodology to the ancillary company that manufactures fuel delivery systems for passenger cars and light trucks to discuss

the efficacy of using PDS approach in a real manufacturing facility.

This paper is organized as follows. In Section 2, we present the sets of system parameters and dispatching rules considered in this study. In Section 3, we first overview a generic inductive learning approach, and subsequently, discuss the refinements developed in this study for the mean tardiness problem. We conclude this section with a discussion of the decision tree generated and its ability to synthesize insights into the problem. In section 4, we show that the superiority of the PDS policy over the conventional scheduling rules is carried to a real system. The major results of this paper are summarized in Section 5.

2. System Descriptors and Dispatching Rules to Consider for an Adaptive Scheduling

2.1 System State Descriptors

The set of important system parameters to be considered is based upon past research, and it includes both system and job characteristics. These are now discussed.

1. *Flow Allowance* : This measures the lead time permitted to any job, and is, therefore, a measure of due date tightness. At any point in time t, the (remaining) flow allowance, f_j of job j is expressed as $f_j = (d_j - t)/p_j$ *where* d_j and p_j, respectively, refer to the due date and the remaining processing time of j.
2. *System Utilization* : From basic queuing theory, it is known that job flow time, and therefore job tardiness as well, depends upon the system utilization.
3. *Contention Factor* : The contention factor c_i of any operation i in any job is the number of alternative machines available for processing it. It is well-known that providing parallel servers can reduce system congestion and job flow time.
4. *Buffer Size* : Many FMSs are tightly constrained in terms of the available buffer space b. When this constraint is binding, and particularly in the presence of job shop-like random material flows, various machines in the system are likely to go through phases of blocking and starving which adversely affect the system performance.
5. *Relative Machine Workloads* : It is well known that bottlenecks impact system performance critically. Raman et al. (1989) show that relative workload distribution can also impact the selection of an appropriate scheduling rule. Two measures are used in this study to determine relative machine workloads.

   $\text{workload}_w = W_{max}/W_{ave}$ measures the ratio of the maximum workload at any machine to the average workload. At any instant, these workloads reflect the remaining processing to be

done on the available jobs. The other measure $workload_{cv}$ is the normalized standard deviation of relative workload, expressed as the ratio of the standard deviation of individual machine workloads W_m to the average machine workload W_{ave}.

6. ***Machine Homogeneity*** : This measures the variability in the number of operations that individual machines can process.

2.2 Dispatching Rules

Dispatching rules differ in how they assign priority indices to waiting jobs. Rules that have been found to be effective in previous studies include i) the Earliest Due Date (EDD) rule (Baker and Bertrand 1982), ii) the Shortest Processing Time (SPT) rule (Baker and Bertrand 1982; Conway et al. 1967), iii) the Modified Job Due Date (MDD) rule (Baker and Kanet 1983; Raman et al. 1989), and iv) the Modified Operation Due Date (MOD) rule (Baker 1984; Raman et al. 1989). While a number of other rules have been proposed in the literature, we have selected four rules that have been found to be effective for the objective of minimizing mean tardiness in the past (Baker 1984).

3. Constructing a Pattern-Directed Scheduling Approach

Given the set of dispatching rules, and the set of system descriptors, the third element of the adaptive scheduling policy, namely the set of transformation rules is developed through inductive learning. In this section, we first review the basic concepts of inductive learning and next recapitulate the construction of the adaptive scheduling policy developed in Shaw et al.(1992).

3.1 Inductive Learning

Inductive learning can be defined as the process of inferring the description (that is, the concept) of a class form the description of individual objects of the class (Shaw et al. 1992). A concept to be learned in scheduling, for example, can be the most appropriate dispatching rule (a class) for a given manufacturing pattern.

Generalization and specialization are essential steps for the inductive learning process. A generalization of an example is a concept definition which describes a set containing that example. In other words, if a concept description G is more general that the concept description F, then the transformation form F to G is called generalization; a transformation form G to F is specialization. For a set of training examples, the generalization process identifies the common

features of these examples and formulates a concept definition describing these features; the specialization process on the other hand, helps restrict the coverage of features for a concept description. Thus, inductive learning can be viewed as the process of making successive iterations of generalizations and specializations on concept descriptions as observed form examples. This process continues until an inductive concept description which is consistent with all the training examples is found. Thus the generalization to specialization relations between concept descriptions provide the basic structure to guide the search in inductive learning.

The input to an inductive learning algorithm consists of three groups: i) A set of positive and negative examples (favorable and unfavorable manufacturing system state descriptions for a given dispatching rule), ii) a set of generalization and other transformation rules, and iii) criteria for successful inference. Each training example consists of two components – a data case consisting of a set of attributes, each with an assigned value (manufacturing state descriptions); and the classification decision made by a domain expert according to the given data case (the best dispatching rule). The output generated by this inductive learning algorithm is a set of decision (mapping) rules consisting of inductive concept definition for each of the classes. Learning programs falling into this category include AQ15 (Michalski 1983), PLS (Rendell 1983) and ID3 (Quinlan 1986). These programs are referred to as similarity-based learning methods.

In this research, we use C4.5(Quinlan 1988) which is a refinement of the ID3 program. The learning process in C4.5 follows a sequence of specialization steps guide by an information entropy function for evaluating class membership. The concept description generated by a learning process can be represented by a decision tree. The nodes of the tree represent conditions set of attribute values; each branch corresponds to a disjunctive normal form expression. From the previous PDS experience (Park 1991), the quality of the decision tree we obtained from inductive learning was relatively satisfactory. Since this paper extends the features and efficacy of dynamic scheduling capability of PDS, we leave the employed inductive learning algorithm C4.5 untouched. Perhaps, in subsequent studies, it is worth while to compare the improved performance of PDS using the knowledge obtained from hybrid of inductive learning and linear or non-linear discriminant analysis methods generating general hyperplanes.

3.2 Rule Refinement Process

The rule refinement stage provides a control mechanism for the purpose of insuring an acceptable scheduling performance level. This stage monitors the quality of the schedules generated at the second stage by comparing its performance with those obtained by repeatedly applying each dispatching rule in D individually, under a variety of scenarios. Higher mean tardiness values

under PDS indicate deficiencies that need correction. These deficiencies could be caused, for example, by not considering a large enough set of training examples. As noted earlier, it is difficult for the set of training examples to be comprehensive in view of the vastness of the system attribute space. Consequently, the heuristic selection rules imbedded in the r-tree and θ-trees are overgeneralized to some extent. If this results in performance degradation, then these trees need to be refined.

Formally, let ε^n denote the set of training examples generated until stage n. An example $e \in \varepsilon$ is tuple $\{c, d\}$ where $c \in C^n \subset M$ represents a pattern , C is the set of all patterns investigated through stage n, and d is the dispatching rule found appropriate for c. Let R^n denote the set of rules imbedded in the r-tree through stage n and let $|R^n| = R_n$. For any rule $r_i \in R^n$, let $\varepsilon_i^n = \{C_i^n, d_i\}$ be the set of supporting training examples, where C_i^n is the set of patterns considered in these examples and d_i is the resulting dispatching rule. The inductive learning algorithm insures that

$\cup C_i^n = C$, and $\cap C_i^n = \{\}$

Note, however, that as a result of the imbedded generalization in the algorithm, the system attribute state space covered by r_i is M_n^i such that

$C_i^n \subset M_n^i$, *and* $\cup M_n^i = M$.

If C_i^n is not a complete representation of M_n^i, then overgeneralization occurs resulting in a prediction error.

The rule refinement procedure identifies all such instances of incomplete representations, and augments the decision tree by generating additional rules appropriately. The metric used for rule refinement is ***prediction accuracy*** α that measures the proportion of testing instances in which the scheduling rule selected by the r-tree turns out be the one that performs the best among all rules in D when implemented individually. On a random sample of test problems, if α is found to be less than h, the prespecified target prediction accuracy level, then the learned rules are refined following a three-step process. First, the deficient rules in the r-tree are identified; next, additional training examples are generated in order to specifically address preconditions manifest in the deficient rules. In the final step, the inductive learning algorithm is employed to update the tree on the basis of the additional information provided by these examples. The process of generating the testing instances, evaluating α on these instances, and refining the tree is carried out iteratively until the desired prediction accuracy is achieved. This method is formally stated below; a detailed description follows subsequently.

Algorithm RefineRule

Step 1. Initialization: Set $n = 1$, R^1 at the initial decision tree, and C_n^i the patterns considered for constructing this tree. Go to Step 2.

Step 2. Testing Example Generation: a) For each $r_i \in R^n$, determine C_i^n, and $P_i^n = M_n^i - C_i^n$. Determine the set of testing patterns $T^n \subset (P_i^n$ such that $|T^n| = k$.

b) Generate set S^n consisting of k testing examples obtained by performing simulation runs on T^n for each dispatching rule in D. Go to Step 3.

Step 3. Termination: a) Determine the prediction accuracy α by using R^n on S^n.

b) IF $\alpha \leq h$, THEN stop or go to Step 2 for evaluating next testing set. ELSE, go to Step 4.

Step 4. Iteration: a) For each $r_i \in R^n$, determine the set of testing patterns $W_i^n \subset T_i^n$ for which d_i is not found to be the best dispatching rule. Let $N_i^n = \{(W_{i1}^n, d_{i1}), \cdots\cdots, (W_{iwi}^n, d_{iwi})\}$ be the corresponding set of testing examples where W_{il}^n denotes the subset of patterns for which the best dispatching rule was found to be d_{il}, $l = 1, \cdots\cdots, W_i$, $d_{il} \neq d_i$, and W_i is the number of such subsets.

b) For each $r_i \in R^n$, generate a set of rules R_i^n using the inductive learning algorithm with $\varepsilon_i^n \cup N_i^n$ as the set of training examples.

c) Identify suspicious regions $Q_i^n = C_i^n \cap (\cup C_q^n$, *and* $r_q \in R_i^n$, $d_q \neq d_i)$.

Step 5. Additional Training Example Generation: Generate set A_n consisting of a additional training examples obtained by performing simulation runs on Q^n for each dispatching rule in D.

Step 6. Refined Rule Generation: Generate a set of rules R_i^{n+1} using the inductive learning algorithm with $\varepsilon^n \cup A^n$ as the set of training examples, and go to Step 3.

The testing examples used in Step 2 of this procedure considers the subspace P that is not covered by the training examples in order to check the occurrence of any incomplete representations. Such an occurrence is indicated if, for any subspace, the dispatching rule found dominant in the testing examples is different from the one indicated by the current r-tree. If the proportion of such occurrence is higher than the acceptable level, this subspace is divided in Step 4 into smaller subspace; each of these smaller subspace being dominated in the testing examples by a different dispatching rule. The r-tree is accordingly updated, and at the next iteration, the testing examples used address the possibility of overgeneralization for the new rules added at the current iteration. The attribute subspace covered by the testing examples consequently reduces at each iteration, thereby guaranteeing the convergence of this algorithm (also see Politakis and Weiss 1984, Sammut and Banerji 1986, Tecuci and Kodratoff 1990, and Pazzani et al. 1991).

4. Experimental Study

The PDS system is implemented on VAX 8800 VMS environment, IBM 3090 CMS environment, and DecStation 5000/200 Ultrix environment. Simulation module was written in SLAM II simulation language. Inductive learning module was written in C (at this moment, learning module is off-line). Each simulation run and knowledge induction takes less than 2 minutes.

To verify the effectiveness of the PDS approach, we have applied the PDS system in an auto ancillary company that manufactures fuel delivery systems for passenger cars and light trucks. This facility produces 41 different products on two identical manufacturing lines. (See Hausman, Lee and Masri 1987 for a detailed description of the system.) Although the operations are driven by a monthly production schedule, there are frequent changes in this schedule that result in expediting some orders and delaying the due dates of other orders.

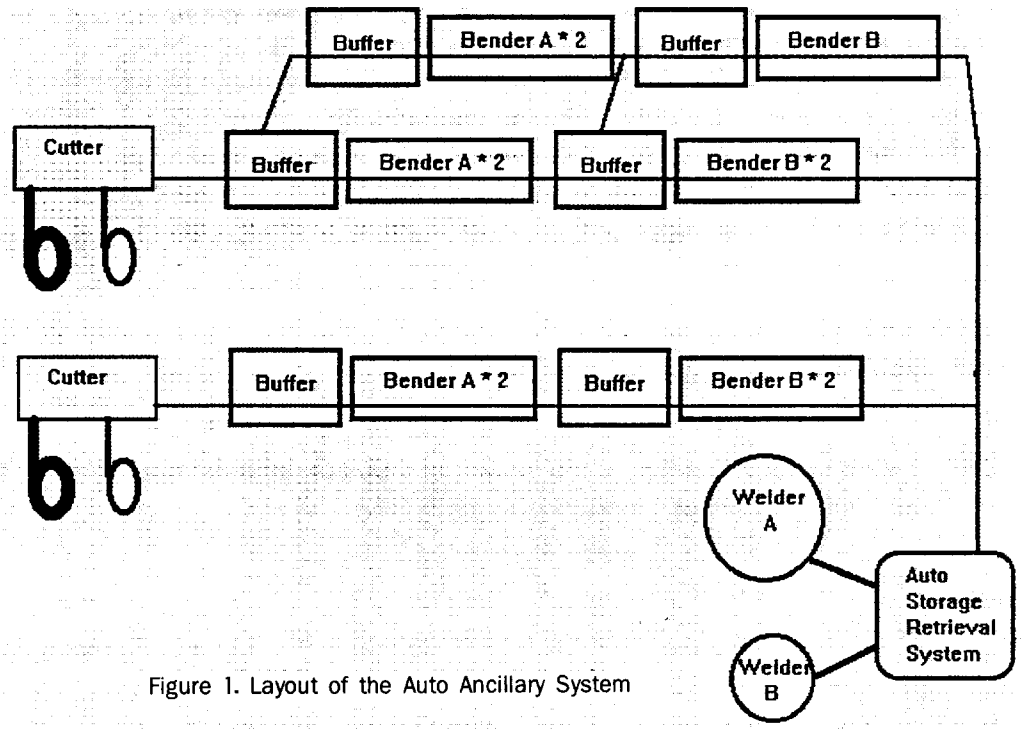

Figure 1. Layout of the Auto Ancillary System

Figure 1 gives the layout of the facility. Each manufacturing line consist of two stages tube cutting and tube forming. The processing time for the tube cutting operation is 36 seconds (The actual processing time values are suppressed; the numbers shown here are representative of the actual figures.) for all products at each of the two cutters available on each line. Tube forming is

done on one of 11 identical machines available; the tube forming time depends upon the tube geometry and it varies from 60 seconds to 105 seconds. Formed tubes are subsequently sent to the welding stage comprising two welding machines. The welding time is 30 seconds for small tubes and 42 seconds for large ones.

Depending upon the composition of the orders, the system utilization varies from 30% to 80%. A range of due date tightness is achieved by allowing flow allowance factor to vary between 2 and 12. The size of buffer available at each bender is 120 while it is virtually unrestricted for cutters and welders. Machine workloads and the average contention factor at any point in time depend upon the composition of jobs present in the system at that instant. Based on the processing times and the past demand data, the workloadw is seen to vary between 1.2 and 3.0, while workloadcv varies between 0.1 and 0.8. The contention factor for the overall system ranges between 1.9 and 2.6.

For the purpose of this set of experiments, the threshold level of prediction accuracy h was set to 0.9. The θ-tree turns out to be a singleton with θ=0.2 performing the best in 20 of 21 training examples. The r-tree generated initially from 37 training examples comprised 6 selection rules as shown in Figure 2. When tested on 18 scenarios, it yielded a prediction accuracy α of 77.8%. At this stage, 12 new training examples were added for the purpose of rule refinement. Augmenting the r-tree appropriately resulted in 8 selection rules and α=0.944; since the desired prediction accuracy was achieved, no further refinement was carried out. The final r-tree is shown in Figure 3.

Figure 2: Initial r-tree for the Real System

```
IF(contention-factor. GT.2.139) GO TO 10
    IF(workload_cv. GT.0.298) GO TO 20
        r = SPT;
        20: IF(system-utilization. GT.64.074) GO TO 30
            r = MDD;
            30: IF(flow-allowance. GT.4.0) GO TO 40
                r = EDD;
                40: r= MDD;
    10: IF(system-utilization. GT. 71.234) GO TO 50
        r = MOD;
        50: r = EDD;
```

Figure 3: The r-tree after Rule Refinement for the Real System

```
IF(contention-factor. GT.2.139) GO TO 10
    IF(workload_cv. GT.0.252) GO TO 20
        r = SPT;
        20: IF(flow-allowance. LE.4.0) GO TO 30
            r = MDD;
            30: IF(flow-allowance. LE.3.0) GO TO 40
                r = EDD;
                40: IF(system-utilization. GT.64.8) GO TO 50
                    r = MDD;
                    50: IF(workload_cv. GT. 0.317) GO TO 60
                        r= MDD;
                        60: r = EDD;
10: IF(system-utilization. GT.71.234) GO TO 70
    r = MOD;
    70: r = EDD;
```

Two sets of experiments were conducted to evaluate PDS relative to other scheduling rules. The first set evaluated PDS under stationary operating conditions. In order to capture the impact of frequent changes in order due dates, the second set of experiments allowed random shifts in the mean flow allowance value. The experimental results for these two set are shown in Tables 1 and 2, respectively.

Table 1 confirms the overall superiority of PDS observed in the earlier experiments for the stationary case. Relative to the BEST rule, PDS is found to be better in 6 of 21 problems, equal in 11 problems, and worse in 1 problem. On average, it improves the mean tardiness values obtained under STP by 16.3%, EDD by 4.6%, MDD by 1.7%, MOD by 20.5%, and BEST by 0.8%.

Table 2 shows that PDS continues to be increasingly superior in the nonstationary case. It is better than the BEST rule in all 11 cases. Relative to PDS, SPT yields 4.6% higher than mean tardiness, and the corresponding figures for EDD, MDD, MOD and BEST are 6.0%, 7.2%, 6.2% and 1.3%, respectively.

Table 1. The PDS Performance for the Real System: The Stationary Case

PDS	SPT	EDD	MOD	MDD	BEST	D
1.105	1.46	1.806	1.703	1.806	1.46	24.32
1.286	1.799	2.107	2.058	2.107	1.799	28.52
1.19	1.557	1.945	1.865	1.945	1.557	23.57
0.9981	1.207	1.582	1.441	1.582	1.207	17.31
0.8601	1.073	1.406	1.281	1.406	1.073	19.84
1.213	1.648	1.983	1.91	1.983	1.648	26.40
5.062	7.679	5.062	8.006	5.062	5.062	0.00
28.14	29.78	28.14	30.39	28.14	28.14	0.00
5.28	8.308	5.28	8.647	5.28	5.28	0.00
26.95	28.04	26.95	28.61	26.95	26.95	0.00
5.096	7.325	5.096	7.814	5.096	5.096	0.00
25.74	27.6	25.74	28.13	25.74	25.74	0.00
4.751	6.834	4.751	7.073	4.751	4.751	0.00
38.2	45.48	45.82	47.42	38.2	38.2	0.00
27.44	28.61	27.44	29.02	27.44	27.44	0.00
8.43	11.3	8.43	11.74	8.615	8.43	0.00
23.64	25.98	23.64	26.43	23.64	0.00	
4.154	5.742	4.154	5.898	4.154	4.154	0.00
21.71	24.9	21.71	25.3	21.71	21.71	0.00
3.764	5.104	3.764	5.242	3.764	3.764	0.00
19.53	24.59	19.47	26.62	19.53	19.47	−0.31

***D = (BEST−PDS) /BEST *100

5. Conclusion

This paper develops an adaptive scheduling policy for dynamic manufacturing systems. The main feature of this policy is that it tailors the dispatching rule to be used at a given point in time to the prevailing state of the system. The rule selection logic is imbedded in a decision tree that is generated by applying an inductive learning algorithm on a set of training examples.

Experimental studies dealing with a stationary FMS confirm the superiority of the suggested PDS approach over the alternative approach that involves the repeated application of a single dispatching rule previously found in Shaw et al.(1992). In addition, this study shows that, for a real system, the relative performance improves further when the system is nonstationary. In particular, PDS performs better when there are frequent disruptions, and when disruptions are caused

Table 2. The PDS Performance for the Real System: The Non-Stationary Case

flow allow -ance	magn -itude	inter -val	frequ -ency	PDS	SPT	EDD	MOD	MDD	BEST	D
4	-2	400	1	16.14	16.24	17.47	16.82	17.76	16.24	0.62
4	-2	800	1	18.21	18.22	19.53	18.83	19.87	18.22	0.05
4	-1	400	1	4.105	4.436	4.448	4.775	4.448	4.436	7.46
4	-2	400	2	16.17	16.24	17.62	16.8	17.77	16.24	0.43
4	-2	400	4	3.552	3.793	4.284	4.031	4.817	3.793	6.35
4	-1	400	2	4.182	4.432	4.493	4.768	4.493	4.432	5.64
4	-1	400	4	1.799	2.157	2.49	2.376	2.49	2.157	16.60
6	-4	400	1	0.3461	1.054	0.3461	0.7161	0.3461	0.3461	0.00
6	-4	800	1	17.2	17.63	17.24	17.26	17.2	17.2	0.00
6	-4	1600	1	21.59	23.39	21.64	23.04	21.59	21.59	0.00
6	-3	1600	1	0.8387	1.31	0.8387	1.212	0.8387	0.8387	0.00

by the sudden introduction of urgent jobs, one of the most common sources of disruptions in most manufacturing systems.

From an operational perspective, the most important characteristics of the PDS approach are its ability to incorporate the idiosyncratic characteristics of the given system into the dispatching rule selection process, and its ability to refine itself incrementally on a continuing basis. The first characteristic highlights the fact that the decision trees are system specific, and emphasize the need, on a part of the decision maker, to pursue a scheduling policy(such as PDS) instead of using a single dispatching rule. The second characteristic insures that decision trees are self correcting and current. As discussed earlier, all selection rules imbedded in the tree are overgeneralized to some extent. If this results in inferior performance, then the trees need to be augmented with additional rules. Furthermore, while the set of system parameters and the set of dispatching rules considered for generating the training examples need to be comprehensive, they change over time as new parameters are added, and new dispatching rules become available. The built-in rule refinement procedure, when used in conjunction with periodic a posteriori comparisons of PDS with other dispatching rules, insures that the selection rule base is maintained efficiently.

REFERENCES

[1] Baker, K.R., "Sequencing Rules and Due-Date Assignments in a Job Shop," Management Science, Vol. 30, No. 9 (1984), pp. 1093-1104.

[2] Baker, K.R., and J.M. Bertrand, "A Dynamic Priority Rule for Sequencing Against Due Dates," Journal of Operations Management, Vol. 3 (1982), pp. 37-42.

[3] Baker, K.R., and Kanet, J.J., "Job Shop Scheduling with Modified Due Dates," Journal of Operations Management, Vol. 4, No. 1 (1983), pp. 11-22.

[4] Conway, R.W., Maxwell, W.L., and Miller, L.W., Theory of Scheduling, Addison-Wesley, Reading, MA, 1967.

[5] Hausman, W.H., Lee, H.L., and Masri, S.M., "Dynamic Production Scheduling for Fuel Sender Manufacturing," Working Paper, Department of Industrial Engineering Management, Stanford University, Palo Alto, CA., 1987.

[6] Michalski, R.S., "A Theory and Methodology of Inductive Learning," In R. Michalski, J. Carbonell, and T. Mitchell, (Eds.), Machine Learning: An Artificial Intelligence Approach, Tioga, Palo Alto, CA, 1983.

[7] Park, S.C., "Applying Machine Learning to the Design of Decision Support Systems for Intelligent Manufacturing," Ph. D. Dissertation, Department of Business Administration, University of Illinois-Urbana & Champaign, 1991.

[8] Pazanni, M.J., Brunk, C., and Silverstein, G., "A Knowledge-intensive Approach to Learning Relational Concepts," Proceedings of the Eighth International Workshop on Machine Learning, pp. 432-436, Morgan Kaufmann, San Mateo, CA, 1991.

[9] Politakis, P., and Weiss, S., "Using Empirical Analysis to Refine System Knowledge Bases," Artificial Intelligence, Vol. 22 (1984), pp. 23-48.

[10] Quinlan, J.R., "Induction of Decision Trees," Machine Learning, Vol. 1(1986), pp. 81-106.

[11] Quinlan, J.R., "Decision Trees and Multi-Valued Attributes," Machine Learning, Vol. 11 (1988), pp. 305-318.

[12] Raman, N., Talbot, F.B., and Rachamadugu, R.V., "Due Date Based Scheduling in a General Flexible Manufacturing System," Journal of Operations Management, Vol. 8, No. 2 (1989), pp. 115-132.

[13] Rendell, L., "A New Basis for State-Space Learning Systems and a Successful Implementation," Artificial Intelligence, Vol. 1, No. 2 (1983), pp. 177-226.

[14] Sammut, C., and Banerji, R., "Learning Concepts by Asking Questions," In R. Michalski, J. Carbonell, and T. Mitchell, (Eds.), Machine Learning: An Artificial Intelligence Approach, pp. 167-192, Tioga, Palo Alto, CA, 1983.

[15] Shaw, M.J., Raman, N., and Park, S.C., "Intelligent Scheduling with Machine Learning Capabilities: The Induction of Scheduling Knowledge," IIE Transactions, Vol. 24, No. 2 (1992), pp. 156-168.

[16] Tecuci, G., and Kodratoff, Y., "Apprenticeship Learning in Non-homogeneous Domain

Theories, In Kodratoff, Y., and Michalski R. S. (Eds.) Machine Learning Vol. III, pp. 514-551, Morgan Kaufmann, San Mateo, CA, 1990.