◈ Research Papers

A Heuristic Scheduling Algorithm for Minimizing Number of Tardy Jobs

Park, Chi Yeon*
Yoo, Wook Sung**

Abstract

Since on-time delivery is becoming increasingly important in today's competitive markets, the development of effective scheduling methodologies to meet customer needs through on time delivery is of vital importance for the well-being of companies. The objective of this study is to develope a general heuristic algorithm for scheduling both the static and dynamic problems under certain on-time delivery criteria. For the problems with due-date related, we present insights for minimizing the number of tardy jobs and propose a heuristic scheduling algorithm. Experimental results show that the proposed algorithm performs significantly better than other dispatching rules within reasonable computation time, specially in the environment of strong competition for resources.

1. Introduction

Job shop scheduling is a resource allocation problem subject to allocation and sequencing constraints. Each job may consist of several subtasks or operations which can be interrelated by precedence restrictions. In the general job shop scheduling problem, n jobs are processed by m machines under given assumptions, and the problem is to determine the sequence of n jobs on each machine that optimizes a given objective function. The objective of the classical job shop is to minimize the time needed to process all operations.

Since many scheduling problems encountered in real world are far more complex than the classical job shop problems, the assumptions of the classical job shop are often violated in practice. The decision of which performance measure to use usually depends on one's perception of its most important goals. For example, if a company regards maximum utilization as the most important goal, the choice would be to minimize the makespan. On the other hand, if the company decides to minimize customer complaints about late deliveries, the choice would be due-date related performance measures. The problems studied in this research are examples of case where the performance measure is not makespan but the due-date related which reflects

* Dept. of Computer Science & Engineering, Kwandong University
** Dept. of Computer & Information Science, Gannon University
This research was supported by Kwandong University Research Grant 1999.

the recent emphasis given to customer satisfaction in many industries[1,2]. The objective is to sequence the jobs so that the number of jobs completed after their due-dates is minimized.

In section 2, the related works are briefly presented. Section 3 describes the principle of the proposed algorithm, General Algorithm Framework. The characteristics and optimal properties of four typical categories of problems are discussed in detail along with the steps of the algorithm. Experimental setup is made and simulation results are carried out in section 4. Finally, the conclusion follows in section 5.

2. Related works

We will now define the following notation, which will be consistently used throughout this paper. Additional notation will be defined as needed.

. n : number of jobs
. m : number of machines
. d_j : due-date of job j
. p_j : processing time of job j
. r_j : release time of job j at which j is ready for processing
. C_j : completion time of job j
. L_j : lateness of job j, which is $C_j - d_j$
. T_j : tardiness of job j, where $T_j=1$ if $C_j>d_j$, $T_j=0$ otherwise
. $C_{max} = \max_{1\le j\le n}\{C_j\}$
. $L_{max} = \max_{1\le j\le n}\{L_j\}$
. $N_T = \sum_{j=1}^{n} T_j$

In addition, we will also use the three-field notation of Graham et al.[3], $\alpha/\beta/\gamma$, to represent each problem where

. first field(α) specifies the machine environment
. second field(β) indicates job characteristics
. third field(γ) refers to the optimality criterion chosen

For example, $m/r_j,prec,s_{ij}/\Sigma T_j$ represents the problem of sequencing m multiple machines in the presence of non-simultaneous arrival, precedence constraints and sequence-dependent setup times to minimize the number of tardy jobs.

The problem of minimizing the number of tardy jobs(ΣT_j) is sometimes more difficult to optimize than makespan(C_{max}) or maximum lateness(L_{max}). The $1//T_j$ problem[1] is solved by the O($n$log$n$) algorithm of Moore[4], usually called Hodgson's algorithm. Sturm[5] provided a simpler proof of optimality for Hodgson's algorithm using mathematical induction. An extension of Moore's algorithm for the case in which certain jobs have to be on time was given by Sidney[6].

Kise et al.[7] considered the problem where setup and processing times are independent with normally distributed random variables, and showed that the problem can be solvable if ready times and due-dates are agreeable, ie., $r_i<r_j$ implies $d_i<d_j$ [23]. Lawler[8] generalized Hodgson's algorithm to the weighted number of tardy job problem, $1//\Sigma W_jT_j$, with the case of agreeable weights, i.e., $p_i<p_j$ implies $w_i<w_j$. Sahni[9] gave dynamic programming algorithms for the $1//\Sigma W_jT_j$ problem. Later, Villarrel and Bulfin[10] provided a branch and bound algorithm with extensive computational results based on two lower bounding

procedures and a dominance theorem.

Potts and Van Wassenhove[11] also proposed a branch and bound algorithm which uses problem reductions derived from the knapsack problem and dominance relations. Monma and Potts[12] examined the number of tardy job with setup time problem in the case of batch setups. These authors proved a result concerning the structure of the optimal solutions and provided a dynamic programming algorithm that is polynomial in the number of jobs but exponential in the number of batches. Uzsoy et al.[13] developed an algorithm based on Moore's algorithm for minimizing the number of tardy jobs on a single workcenter with precedence constraints and sequence dependent setup times.

Uzsoy et al.[14] showed the worst-case behavior of this algorithm for the case without precedence constraints. Lawler et al.[15] studied preemptive scheduling problems on a number of parallel machines and they developed an $O(n^3)$ algorithm for the special case of two uniform machines. Li[16] presented a heuristic, which is a generalization of Kise et al's algorithm for scheduling jobs on parallel machines with agreeable due-dates.

3. General Algorithm Framework(GAF)

This study focuses on the performance measure of minimization of the number of tardy jobs, one of the regular measures. For the static problem, Moore's algorithm optimally solves in polynomial time. However, in the environment of dynamic job arrival situations, this problem turns to be NP hard. Thus, the problem needs to be solved in some other ways in reasonable computation time. We propose a heuristic method to tackle the problem in general. The algorithm can be stated as follows.

Initialize temporary sequences S, S′, S″, and R to ∅.

Step 1: Remove the jobs which cannot be completed on time regardless of the order and move them to R.

Step 2: Obtain S for the remaining jobs by EDD(Earliest due-date) rule.

Step 3: Append all jobs in S′ to S. If there is any tardy job in S, remove the jobs just after the first tardy job thru the end and replace S′ with them. Otherwise, go to Step 6.

Step 4:

4.1 For the given sequence up to the first tardy job, reschedule the jobs in S to see if there exists a new sequence S″ with no tardy job.

4.2 If there is no new sequence without a tardy job, go to Step 5.

4.3 Otherwise, replace S with the new sequence S″, and go to Step 3. (If there are more than one new sequence found, choose the one having minimal completion time.)

Step 5 :

5.1 Among the jobs in S, find the candidate job whose removal allows the rest of jobs in S to be completed on time.

5.2 If multiple candidates are found, choose the job whose removal results in minimal completion time for the last job in the subsequence. Remove the job from S and add it to R. Go to Step 3.

Step 6 : Create the final sequence by adding all jobs in R to the end of S.

The main idea of this algorithm is to maintain the best possible subsequence up to the first tardy job in a given sequence. It includes the steps to reflect the insights found for the number of tardy job measures, step 4 and 5. In step 4, since the subsequence up to the first tardy job may not be an optimal sequence in complex problems, the subsequence needs to be rescheduled before a job is considered to be removed where resequencing methods are discussed later. The idea of step 5 is that if one job has to be taken away for the improvement of result, it removes the job which results in minimal completion time for the subsequence. Following sections describe each step in detail for the problems of 4 typical categories.

3.1 Problems of $1 / / \Sigma T_j$

For the simple problem with no setup time, no precedence relation, and no dynamic job arrival, the General Algorithm Framework works the same as Hodgon's algorithm, since Lemma 2-1 holds and removal of the longest processing job guarantees the minimal completion time for the subsequence.

Lemma 2-1 : For the problem of $1/ / \Sigma T_j$, the sequence generated according to the EDD(Earliest Due Date) rule alway guarantees the optimal subsequence up to the first tardy job. In other words, there is no subsequence without any tardy job for the given subsequence when generated by EDD rule.

Proof : Let S be the sequence with EDD order, and job i is the first tardy job. Then $d_i \geq d_k$, $\forall k$ = 1, 2, ..., i-1, and $d_i < P$, where d_i is due-date for i, and P is total processing time of the jobs in the subsequence. Reschedule the subsequence S, and let S' be any new subsequence in which job i is not tardy, and job j be the last job in S'. Because the total processing time, P, is constant, the relation $C_j = P > d_i \geq d_j$ holds. Therefore job j should be tardy, which means any subsequence has at least one tardy job. Hence, the subsequence up to the first tardy job using EDD rule is an optimal subsequence.

3.2 Problems of $1/r_j/\Sigma T_j$

Following is the description of the General Algorithm Framework for the problems with dynamic job arrival, no setup times, and no precedence relations.

In step 1, find the jobs to remove by comparing the due-date(d_j) to the sum of ready time(r_j) and process time(p_j) of job j, i.e., remove job j if $d_j < r_j + p_j$. In step 2 and 3, the due-dates of all jobs are not simply able to be compared, since not all jobs are ready at the same time. In such case, the J-EDD(Job with the earliest due date) rule is applied : whenever a machine becomes available, the job with the earliest due-date is first chosen among the jobs awaiting processing.

A feasible sequence can be obtained through step 2 and 3, but the subsequence using this rule may not be optimal in dynamic arrival. A simple example illustrates such case.

job(i)	release time(r_i)	processing time(p_i)	due-date(d_i)
1	0	5	10
2	0	5	15
3	0	5	25
4	14	5	19

For the above instance, the following candidate sequence could be obtained after step 2 (job 4 is not ready when job 2 completed at time 10).

job1 (C_1:5)	job2 (C_2:10)	job3 (C_3:15)	job4 (C_4:20)

Job 4 is found to be the first tardy job. If step 4 in the algorithm is not performed, we would go to step 5 which forces to remove one job. However, note that another possible sequence of 1-2-4-3 would not include any tardy job if considered.

job1 (C_1:5)	job2 (C_2:10)	no job(14)	job4 (C_4:19)	job3 (C_3:24)

This example shows that we do not always have to remove a job from the jobs up to the first tardy job for the given sequence. It is because the subsequence is not guaranteed to be optimal in dynamic situations. Therefore, the current subsequence needs some efforts before actually removing a job from the sequence. Since checking all possible subsequences in Step 4.1 is another exponential problem to be added, some heuristic methods are considered, Neighborhood Search(NS) algorithm[17] and Rolling Horizon Method(RHM)[18,19]. In our study, due to the advantages of NS over RHM in this specific circumstance, NS procedure is employed and modified to limit to only one iteration : examine the sequences of the n-1 possible pairwise exchanges(between adjacent jobs) in the subsequence and find the one with no tardy job and the least completion time.

If the subsequence from step 4 still has a tardy job, we need to find the job whose removal results in no tardy job in the subsequence. Note that removing either the tardy job itself or the job with largest processing time may result in a poor solution for this problem. Instead, step 5 removes the one whose removal results in minimal C_{max} for the remaining subsequence. This is illustrated in the following example.

job(i)	release time(r_i)	processing time(p_i)	due-date(d_i)
1	0	5	5
2	0	4	10
3	9	3	12
4	11	2	13
5	10	6	18

After step 3, the following sequence is obtained in which the first tardy job is job 4.

J1(C_1:5)	J2(C_2:9)	J3(C_3:12)	J4(C_4:14)	J5(C_5:20)

Clearly, any new subsequence consisting of jobs up to job 4 without any tardy job is not found thru step 4. As a result, one in the subsequence should be removed. Let us consider the each case.

(1) job1 removed

J2(C_2:4)	no job(9)	J3(C_3:12)	J4(C_4:14)

(2) job2 removed

J1(C_1:5)	no job(9)	J3(C_3:12)	J4(C_4:14)

(3) job3 removed

J1(C_1:5)	J2(C_2:9)	no job(11)	J4(C_4:13)

(4) job4 removed

J1(C_1:5)	J2(C_2:9)	J3(C_3:12)

In step 5.1, job 3 and 4 are the candidates for removal since the absence of each one results in no tardy job in the subsequence whereas job 1 or 2 still gives a tardy job. Among the candidates, the one ensuring the minimal completion time, job 4, is actually removed in step 5.2. The subsequence of 1-2-3 is taken and step 3 is followed by the algorithm giving the sequence, 1-2-3-5. This sequence doesn't include any tardy job and exits the whole loop. The final sequence 1-2-3-5-4 has one tardy job, job 4.

Instead, if "the longest processing time job removal strategy" is applied to the same instance, the result is quite different. First, in the subsequence 1-2-3-4, job 1 with the longest one is removed and step 3 gives the sequence of 2-3-4-5 in which the first tardy job is still job 4 :

J2(C_2:4)	no job(9)	J3(C_3:12)	J4(C_4:14)	J5(C_5:20)

Next, in the subsequence of 2-3-4, job 2 is removed resulting in 3-4-5 in which the first tardy job is job 4 again. As the iteration proceeds further, only job 4 remains and the final sequence includes four tardy jobs. It is obvious that "removal of job with the longest processing time" strategy is not valid for dynamic situations and in fact this leads to solutions that are substantially worse than GAF.

Since the dynamic-arrival problem is known to be NP hard[20], General Algorithm Framework based on mainly heuristics may not guarantee a global optimal solution. However, it is also important to see that we can get a better solution, if allowed, by applying GAF more than once using the final solution as a new seed. This is because GAF always produces an equal or better sequence than the seed sequence generated by any algorithm that one would use in Step 2 for the number of tardy job measure. Since the number of tardy job measures is a regular measure, shorter completion time for each job in the previous subset will give better results for the whole sequence. At worst, the algorithm will take out the job which is the tardy job itself in subsequence. If it happened all the way to the last job of the sequence, the sequence will have the same number of tardy job as original seed sequence has.

Now, we are able to analyze the computational complexity of GAF. Step 1 and step 2 require only simple comparisons, $O(n)$, and sorting operations, $O(n\log n)$, respectively. The main computation comes from step 4 and step 5 in the iteration. In step 4, the modified NS method requires maximum of $n-1$ swapping operations for which each rescheduled sequence needs n additions to check the tardy job and the completion time. Step 5 consists of maximum of n candidates in which each case needs $n-1$ additions for computing the completion time. The whole algorithm needs maximum of n iterations since at least one job

is appended to the previous sequence before step 4 is executed. Therefore, the algorithm still has a polynomial computational complexity, $O(n^3)$.

3.3. Problems of $1/prec,r_j/\Sigma T_j$

This section describes the GAF for problems with precedence relations. In this problem, each lot may consists of several different operations to go through, which have precedence relations among them. The problems of this category are further classified into static problems and dynamic problems. Since the static problem is a special case of the dynamic problem, $1/prec,r_j=0/\Sigma T_j$, the details of all steps of the algorithm are discussed for the case of dynamic problems.

In Step 1, find the lots to be removed by comparing the minimum amount of total time needed to complete with the remaining time. For the problem without setup time, identify those lots by simply comparing the due-date of each lot with the total processing time of the one. For the problem with sequence independent setup time, the total time needed for a lot is calculated by adding up all setup times and processing times.

In Step 2, obtain a sequence of the operations of lots using J-EDD rule. Step 3 decides the first tardy lot. In this step, we only consider the lots whose last operations are included in the subsequence, "the complete lots". Even if the lots whose last operations are not included, "the incomplete lots", in the current subsequence are already tardy in some cases, the decision is delayed until the last operations of the lots joins to the current sequence.

In steps 4 and 5, the way of finding the lots to remove differs from that of identifying of the first tardy lot in step 3. For the given subsequence, there may exist some incomplete lots. When we consider of removing one, however, we are considering all the lots in the subsequence regardless of their completeness. If the candidate lots consist of both complete and incomplete lots and the removal of the incomplete lot ensures the minimal completion time for the subsequence, follow step 5.2 and remove it. This removal gives the most benefit to the other lots in the rest of the sequence. But, if the removal of the complete lot produces the minimal completion time for the subsequence and there are some incomplete lots included which satisfy the Step 5.1, it is not known if the removal of the complete one will be the best choice for the whole sequence. In this sense, the decision is said to be made locally.

3.4 Problems of $1/r_j,prec,s_{ij}/\Sigma T_j$

In Step 1, for the problem with sequence dependent setup time, it is difficult to identify the jobs to remove in advance because of the setup times. However, this situation can be simplified if we assume that for all operations i, j, and k, the setup time satisfies the triangle inequality, $s_{ik} \leq s_{ij}+s_{jk}$, where s_{ij} is the setup time required to set up operation j after operation i [14]. Under this assumption, the minimum amount of total time needed to complete a lot is the total of the processing time and the setup time based on consecutive operations.

In Step 2, 3 and 4, follow the details of description of problems without setup time except for adding the setup time to every calculation. In Step 5, we need to modify this step a bit for this problem. Since the purpose of this step is to allow the remaining jobs

to start as soon as possible, the setup time should be considered. Let i be the last operation of subsequence, and j be the next operation to i. Instead of removing the job whose removal results in minimal completion time, C, remove the job which results in minimal time of $C+s_{ij}$.

4. Computational Experiments

Experiments were carried out for randomly generated dynamic problems, $1/r_j/\Sigma T_j$, to evaluate the performance of the algorithm developed in this paper. Since no attempt has been made to solve the general case of dynamic problem for these measures, no standard test data set is available. The experimental setup is established excluding the following two meaningless cases.

(1) either the due-dates are so tight that every job is tardy regardless of the algorithm,

(2) or the due-dates are so loose that no job is tardy regardless of the algorithm.

Various experiments were carried out to provide a fair comparison for the performance of the proposed method and several different algorithms.

4.1 Algorithms tested for evaluation

We investigated four algorithms for evaluation, J-EDD, J-EDD with neighborhood search(J-EDD/NS), Branch and Bound, and General Algorithm Framework. J-EDD dispatching rule is designed for due-date related performance measures and has been shown to outperform similar dispatching rules for the Lmax measure[21,22]. We also investigated J-EDD/NS which improves the J-EDD schedule by applying a neighborhood search procedure based on pairwise exchange of adjacent jobs. The GAF is already described in previous section. Note that these three algorithms are essentially heuristic procedures that are not guaranteed to produce a global optimal solution. In order to measure the accuracy of these algorithms, Branch and Bound algorithm is also implemented to obtain an optimal solution for the problems of a moderate size. Brief description of these algorithms are followed.

(1) J-EDD

Whenever a machine becomes available, the job with earliest due-date is chosen among the jobs awaiting processing. This one-pass heuristic has the advantage of easy implementation and usually employed in the environment of relatively loose competition of resources.

(2) J-EDD with Neighborhood Search Algorithm : J-EDD/NS

This algorithm improves the results from J-EDD using pairwise exchange. Following is the description of the algorithm.

Step 1: Obtain an initial sequence S by J-EDD. Set the best sequence S_B to S and the best tardy T_B to the number of tardy jobs in S.

Step 2: Examine each $(n-1)$ possible pairwise exchange in S_B and compute the number of tardy jobs for each. Find the sequence S having the smallest number of tardy jobs T.

Step 3: If $T_B>T$, set T_B and S_B to T and S respectively, and go to Step 2.

Otherwise, stop with the final sequence S_B.

(3) Branch-and-Bound Algorithm

In order to evaluate the performance of the heuristics, a branch and bound algorithm is implemented. Branch and bound algorithm proceeds by partitioning the set of all possible solutions of a problem into subsets and examines each subset. A subset may be fathomed in which if the best possible solution is shown to be worse than one already obtained.

The programs for these algorithms are written in C language on Sun SPARC Station 10 running Unix.

4.2 Experimental Design and Results

The experiments are divided into two parts, the moderate and the large size of problems. Problems of the first part are generated to compare the solution of each algorithm with an optimal solution found by Branch-and-Bound. Since the computational time to find an optimal solution increases tremendously as the problem size grows, this set of experiments are limited to 5-job problems and 10-job problems only. The second part of experiments are generated to compare the quality of solutions of GAF with those of other dispatching rules for bigger size of problems 25, 50, 75, and 100-job problems.

4.2.1. Part 1

A set of 200 sample 5-job and 10-job problems is randomly generated using different random seeds and solved by four different algorithms. Table 1 shows the design of experiment, and results are summarized in Table 2 and 3.

Table 1. Experimental Setup(1)

	Range of values
Number of jobs (n)	5, 10
Release time (r_i)	0 ~ 19
Processing time (p_i)	1 ~ 20
Due-date (d_i)	$r_i + p_i + (0 \sim 39)$

Table 2. Average number of tardy jobs

Problem size	J-EDD	J-EDD/NS	GAF	BB
5 jobs	1.64	1.06	0.83	0.81
10 jobs	7.37	6.30	3.69	3.64

The results indicate that GAF obtains excellent solutions. Experimental results for 5-job and 10-job problems have shown that the proposed method outperforms the J-EDD and J-EDD/NS and never produced poorer results than J-EDD as we expected. In addition, GAF not only performs significantly better than the other dispatching rules but also produces the solutions very close to the optimal solution produced by Branch-and-Bound. We have observed that GAF finds an optimal solution in 98% of trials for the 5-job problems, and in 95% trials for 10-job problems.

We carried out more extensive experiments to see how often each algorithm finds an optimal solution for problems of different degrees of looseness. Table 3 summarize the average percentage of finding an optimal solution for each algorithm. For this experiment,

we have set the same experimental design described in Table 1 except for adding the value "looseness" to the due-dates. The smaller the value of "looseness" is, the tighter the due-date is.

Table 3. Average percentage of finding optimal solutions for 5-job

Looseness	J-EDD	J-EDD/NS	GAF
0	20 %	50 %	98 %
5	26 %	62 %	100 %
10	47 %	83 %	99 %
15	70 %	95 %	100 %
20	88 %	98 %	100 %

These results show that the performance of both J-EDD and J-EDD/NS deteriorates further as the due-dates of the problems become tighter. This can be explained by the ineffectiveness of these dispatching rules in cases with high competition for resources. We had similar results with some more different experimental setups.

In spite of the excellence of the quality of the solutions, GAF requires fairly low computation time. Table 4 shows the average CPU time required to process for each of 5-job and 10-job problems.

Table 4. Average CPU time for each problem

Problem size	J-EDD	J-EDD/NS	GAF	BB
5 jobs	0.17 ms	0.32 ms	0.34 ms	9.17 ms
10 jobs	0.17 ms	0.83 ms	1.16 ms	183036 ms

4.2.2 Part 2

Since the computational efforts for Brand-and-Bound algorithm increases rapidly as problem size increases, the experiments with more than 10 jobs were not carried out with Branch-and-Bound. A set of 400 sample problems are randomly generated and solved by three different algorithms.

Table 5. Experimental setup(2)

	Range of values
Number of jobs (n)	25, 50, 75, 100
Release time (r_i)	0 ~ (10*n -1)
Processing time (p_i)	1 ~ 20
Due-date (d_i)	r_i + p_i + (0 ~ 39)

Table 6. Average number of tardy jobs

Problem size	J-EDD	J-EDD/NS	GAF
25 jobs	13.41	10.19	4.21
50 jobs	32.85	27.99	8.88
75 jobs	55.37	49.37	13.55
100 jobs	75.70	68.67	18.08

Table 5 summarizes the design of experiments. The differences from Table 1 are in the

way the ready times for each job were generated, and the number of jobs for experiments. This method distributes the ready times more uniformly over the whole scheduling time range. Table 6 and 7 show the summary of the experimental results.

Table 7. Average CPU time for each problem

Problem Size	J-EDD	J-EDD/NS	GAF
25 jobs	1.00 ms	7.17 ms	10.17 ms
50 jobs	4.67 ms	33.34 ms	61.32 ms
75 jobs	6.17 ms	85.99 ms	179.65 ms
100 jobs	8.83 ms	172.82 ms	403.81 ms

The results show that GAF is more effective for the bigger size of problems. Notice that other dispatching rules deteriorates further as the size of the problems increases. This can be also explained by the ineffectiveness of other dispatching rules in cases with high competition for capacity. Table 7 summarizes the average CPU time required to process for four different sizes of problems. For 100-job problems, GAF still needs less than half second to solve a problem.

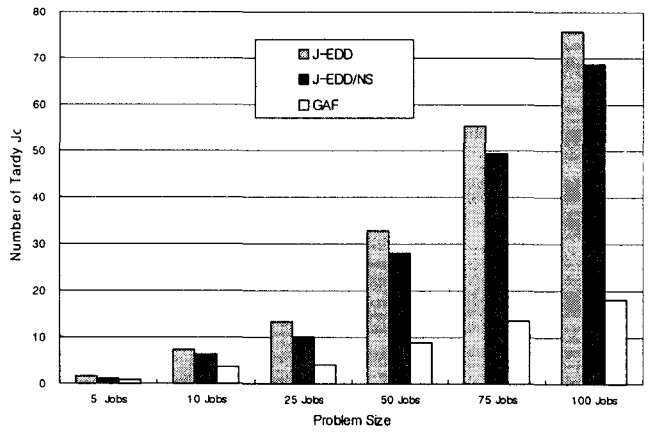

Figure 1. Quality of Solutions

For the comprehensive simulation results, figure 1 depicts the number of tardy jobs remained after applying each algorithm for different size of problems. Extensive computational experiments on these two simulations show that GAF not only consistently outperforms myopic dispatching rules at the expense of very modest increase of computation time, but also find an optimal solution most of times.

5. Conclusion

We developed a General Algorithm Framework to schedule a class of single machine problems under the criterion of the number of tardy jobs. We have revealed that the proposed algorithm yields optimal properties for simple cases, and produces high quality solutions for more complex cases. Experimental results for single machine problems with

dynamic job arrival have shown the method produces near-optimal solution with the computational time comparable to other dispatching rules. In particular, GAF works fast enough to be used in real-time environments. This suggests that GAF is a viable approach in dynamic problems for number of tardy job measure in terms of quality of solution and computational time. Important future directions under the number of tardy job measures would be extension of our results to the more general problems with multiple machines consisting of parallel identical machines and batch processing machines.

References

[1] N.Raman and F.B.Talbot, "The job shop tardiness problem: A decomposition approach", European. Journal. of Operations. Research. Vol. 69, 187-199, 1993.

[2] R.Uzsoy, C.Y.Lee and L.A.Martin-Vega, "A review of production planing and scheduling models in the semiconductor industry", IIE Transaction, Vol. 26, No. 5, 1994.

[3] R.L.Graham, E.L.Lawler, J.K.Lenstra, and R.Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling", Discrete Mathematics Vol. 5, 287-326, 1979.

[4] J.M.Moore, "An n job, One Machine Sequencing Algorithm for Minizing the Number of Late Jobs", Management Science, Vol. 15, No. 1, 1968.

[5] L.B.Strum, "A Simple Optimality Proof of Moore's Sequencing Algorithm", Management Science, Vol. 17, No. 1, 1970.

[6] J.Sidney, "An extension of Moore's due-date algorithm", Symposium on the theory of scheduling and its application, Spainger-Vulag, N.Y., 1973.

[7] H.Kise, T.Ibaraki, and H.Mine, "A solvable case of the one-machine scheduling problem with ready and due times", Operations Research, Vol. 26, 121-126, 1978.

[8] E.L.Lawler, "Sequencing to minimize the weighted number of tardy jobs", Rev.Francaise Automatique, Informatique et Recherche Operationaelle, Vol. 10, 27-33, 1976.

[9] S.Sahni, "Algorithms for scheduling independent tasks", Journal of the ACM, Vol. 23, 116-127, 1991.

[10] F.J.Villarreal, and R.L.Bulfin, "Scheduling semiconductor testing operations: optimization and approximation", Proceedings of the Joint US-German Conference on New Directions for Op. Res. in Manuf., Gaithersburg, MD, July 30-31, 1991.

[11] C.N.Potts, and L.N.Wassenhove, "Algorithms for scheduling a single machine to minimize the weighted number of late jobs", Management Science, Vol. 34, 843-858, 1988.

[12] C.Monma, and C.N.Potts, "On the complexity of scheduling with batch setup times", Operations Research, Vol. 37, 798-804, 1989.

[13] R.Uzsoy, L.A.Martin-Vega, C.Y.Lee, and P.A.Leonard, "Production scheduling algorithms for a semiconductor test facility", IEEE Trans. on Semiconductor Manufacturing, Vol. 4, No. 4, Nov. 1991.

[14] R.Uzsoy, C.Y.Lee, and L.A.Martin-Vega, "Scheduling semiconductor test operations:

Minimizing maximum lateness and number of tardy jobs on a single machine", Naval Research Logistics, Vol. 39, 369-388, 1992.

[15] E.L.Lawler, and C.U.Martel, "Preemptive scheduling of two uniform machines to minimize the number of late jobs", Operations Research, Vol. 37, No. 2, 1989.

[16] C.L.Li, "A heuristic for parallel machine scheduling with agreeable due-dates to minimize the number of late jobs", Computer and Operations Research, Vol. 22, No. 3, 1995.

[17] T.E.Morton, and D.W.Pentico, Heuristic scheduling system, John Wiley & Sons, Inc., 1993.

[18] I.M.Ovacik, and R.Uzsoy, "Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times", Int. Journ. of Prod. Res. Vol. 32, 1243-1263, 1994.

[19] T.E.Morton, R.M.Rachamadugu, and A.Vepsalainen, "Accurate myopic heuristics for tardiness scheduling", #36-83-84, Carnegie Mellon University, Pittsburgh, 1984.

[20] B.J.Lageweg, J.K.Lawler, J.K.Lenstra, and R.Kan, "Computer-Aided complexity classification of deterministic scheduling problems", Mathematisch Centrum, Amsterdam, Rep. BW 138, 1981.

[21] M.Azrak, C.Y.Lee, "Analysis of dispatching policies in semiconductor post-test operations", Journal of Electronics Manufacturing, Vol. 3, 145-157, 1993.

[22] R.Uzsoy, L.K.Church, I.M.Ovacik, and J.Hinchman, "Performance evaluation of dispatching rules for semiconductor testing operations", Journal of Electronics Manufacturing, Vol. 3, 95-105, 1993.

[23] C.L.Li, "A heuristic for parallel machine scheduling with agreeable due-dates to minimize the number of late jobs", Computers and Operations Research, Vol. 22, 1995.