Hybrid job shop scheduling

J.M.J. Schutten
Laboratory of Productions and Operations Management
Department of Mechanical Engineering
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

June 2, 1995

Abstract

We consider the problem of scheduling jobs in a hybrid job shop. We use the term 'hybrid' to indicate that we consider a lot of extensions of the classic job shop, such as transportation times, multiple resources, and setup times. The Shifting Bottleneck procedure can be generalized to deal with those extensions. We test this approach for an assembly shop. In this shop, we study the influence of static and dynamic scheduling, setup times, batch sizes, and the arrival process of the jobs.

1 Introduction

Next to efficiency and quality, delivery performance is very important; cf. Demming [8] and Blackburn [3]. This is particularly true for small batch part manufacturing shops. The planning of those shops is complicated, due to the high product variety and the small batch sizes. Also, most of the jobs are customer specific and occur only once.

The problem of scheduling the jobs in such shops is often modelled as a classic job shop problem. The classic job shop scheduling problem can be described as follows. Given is a shop consisting of m machines $M_1, M_2, \ldots, M_m$. On these machines a set of n jobs $J_1, J_2, \ldots, J_n$ need to be scheduled. Each machine is available from time 0 onwards and can process at most one job at a time. Each job J_j consists of a chain of operations $O_{1j}, O_{2j}, \ldots, O_{n_j,j}$, with n_j the number of operations of job J_j. Operation O_{ij} can only be processed after the completion of operation $O_{i-1,j}$ $(i = 2, \ldots, n_j)$. Operation O_{1j} is available from time 0 onwards. Operation O_{ij} needs uninterrupted processing on machine μ_{ij} during a given non-negative time p_{ij}. The objective is to find a schedule that minimizes the *makespan*, that is, to find a schedule in which the time to process all jobs is minimized.

The plan of this paper is as follows. In the next section, we give some solution approaches for the classic job shop problem. One of them is the Shifting Bottleneck (SB)

procedure of Adams et al. [1]. One of the nice features of this SB procedure is that it can be extended to deal with practical side constraints, such as transportation times, parallel machines at several stages, and so on. In Section 3, we discuss possible extensions of this procedure. In Section 4, we test this approach in an assembly shop in which setup times may occur. Finally, in Section 5, we end with some conclusions.

2 Solution approaches

Each instance of the classic job shop problem can be represented by a disjunctive graph G. For each operation O_{ij}, G has a node v_{ij} with weight p_{ij}. G also has also two auxiliary nodes s and t. For each pair of consecutive operations O_{ij} and $O_{i+1,j}$, G has an conjunctive arc $(v_{ij}, v_{i+1,j})$ $(i = 1, \ldots, n_j)$. Moreover, there is an arc from s to J_{1j} and an arc from $J_{n_j,j}$ to t for each job J_j. The weights of all conjunctive arcs are 0. Between every operation O_{ij} and O_{kl} that must be processed on the same machine, G has a disjunctive edge with weight 0. A feasible schedule is found by orienting all disjunctive edges such that G contains no directed cycle.

Table 1 gives an instances with 3 machines and 3 jobs. Each job consists of 3 operations. Figure 1 gives the graph representing this instance. Figure 2 gives a representation of the

J_j	μ_{1j}	μ_{2j}	μ_{3j}	p_{1j}	p_{2j}	p_{3j}
J_1	M_1	M_3	M_2	4	7	6
J_2	M_2	M_1	M_3	3	5	8
J_3	M_3	M_2	M_1	2	6	7

Table 1: Data for example instance.

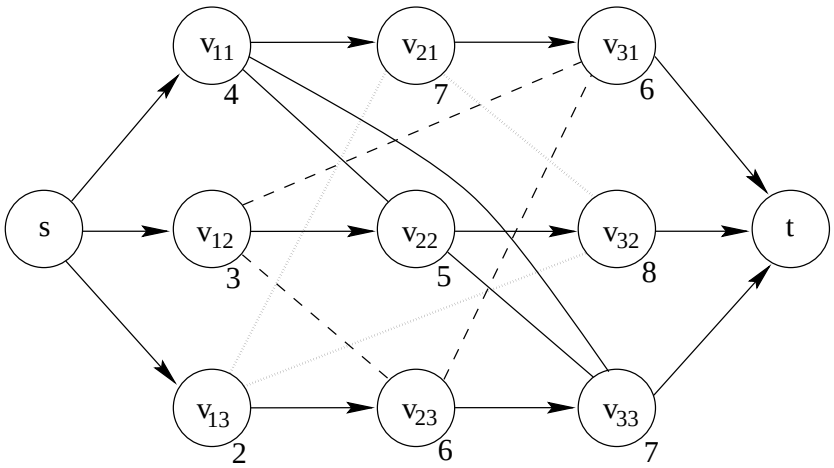

Figure 1: Graph representing example instance.

feasible schedule with $O_{11} - O_{22} - O_{33}$ the sequence on machine M_1, $O_{12} - O_{23} - O_{31}$

the sequence on machine M_2, and $O_{13} - O_{21} - O_{32}$ the sequence on machine M_3. For convenience, we left out the arcs that are induced by transitivity; for example, we left out the arc (v_{11}, v_{33}).

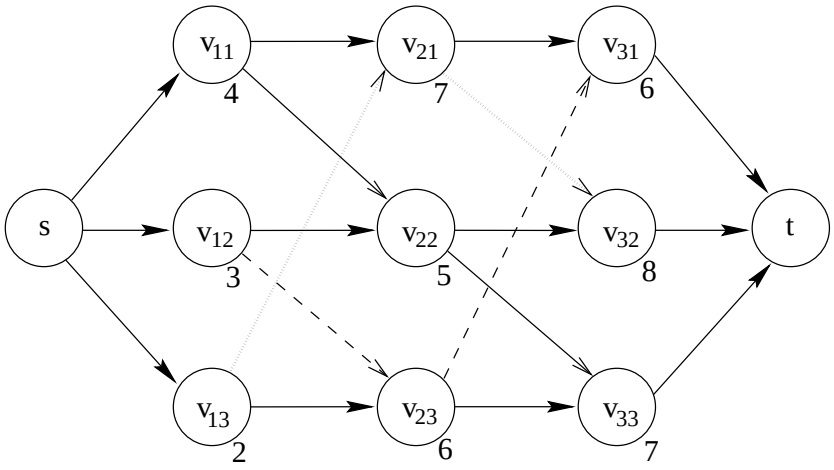

Figure 2: Graph representing a feasible solution.

The classic job shop problem is one of the hardest problems to solve in combinatorics. For example, it took 25 years to solve a problem with only 10 jobs and 10 machines proposed by Fisher and Thompson [9]. Several authors propose branch-and-bound algorithms to solve the problem; cf. Carlier and Pinson [6]. Due to the hardness of the problem, several heuristics have also been proposed. One of them is the Shifting Bottleneck (SB) procedure of Adams et al. [1]. The SB procedure is an intuitive algorithm that decomposes the problem of scheduling a job shop into problems of scheduling single machines. It focuses on bottleneck machines and tries to schedule those machines as good as possible. It schedules the machines one by one. The method heavily relies on calculation of longest weighted paths in graph G discussed above:

- the longest weighted path from node s to a node v_{ij} gives the earliest possible starting time of operation O_{ij}, that is, it gives a release date r_{ij} for operation O_{ij};
- the longest weighted path from node v_{ij} to node t gives the minimum time the shop needs to process all jobs after the completion of operation O_{ij}, that is, it gives a run-out time q_{ij} for operation O_{ij};
- if all machines are scheduled, then the longest weighted path from s to t gives the value of the makespan of this schedule.

The method starts by removing all disjunctive edges from G and by labeling all machines as non-bottleneck machines. Then, longest paths are computed. All machines are scheduled separately using the information gained by the longest path calculations. This means that we need to solve m single-machine scheduling problems in which the jobs have release dates

and run-out times. This can be done with an algorithm of Carlier [4]. The machine with the largest resulting makespan is labelled as a bottleneck machine. The schedule of this machine is fixed by adding the arcs representing the schedule of this machine to G. Now, longest paths are recomputed and the non-bottleneck machines are scheduled using the newly computed release dates and run-out times. The machine with the largest resulting makespan is labelled as a bottleneck machine. The bottleneck machines are now optimized to each other in a special bottleneck optimization step. G is changed, such that it contains all conjunctive arcs and the arcs representing the, possibly changed, schedules of the bottleneck machines. Longest paths are again computed, the non-bottleneck machines are scheduled, and so on. This process continues until all machines are labelled as bottleneck machines.

3 Extensions of the SB procedure

A very nice property of the SB procedure is that it can be generalized to deal with more practical problems. Below, we discuss possible extensions of the classic job shop problem and how we model this.

Release and due dates: The SB procedure assumes that all jobs are available for processing at time 0. Most of the times in practice, however, jobs have different release dates. Suppose that job J_j has release date r_j. If we give the arc (s, v_{1j}) weight r_j, then the longest weighted path from s to v_{ij} is at least r_j. Thus, we ensure that $r_{1j} \geq r_j$, that is, the first operation of this job does not start before the release date of the job. If the jobs have due dates, then the makespan criterium is not so appropriate. We denote the due date of job J_j by d_j. If we give arc $(v_{n_j,j}, t)$ weight $-d_j$, then the longest weighted path from s to t gives the value of the maximum lateness of the current schedule.

Transportation times: In practice, a lot of times it is impossible to start operation O_{ij} immediately after the completion of operation $O_{i-1,j}$, because the job must be transported from one machine to another. We model this by giving arc $(v_{i-1,j}, v_{ij})$ a weight which is equal to the transportation time.

Transportation batch differs from production batch: A job may be an order to produce a whole batch of a product instead of just one product. Then, it may be possible to split an operation O_{ij} on this batch into subbatches. After the completion of a subbatch, it is transported to the next machine in order to decrease the flow time. This subbatch is called a transportation batch. Suppose there are b equal subbatches, then we model this situation by giving the arc $(v_{ij}, v_{i+1,j})$ weight $-\frac{b-1}{b} \cdot \min\{p_{ij}, p_{i+1,j}\}$.

Parallel machines: In the classic job shop every operation must be processed by a given machine. In practice, it might be possible to process an operation by a group of machines. Carlier [5] gives an algorithm that assigns every operation to one of the

machines in the group and that schedules the operations assigned to a machine. Denote the number of parallel machines in the group by k. A schedule for this parallel machine group is then represented by k chains of arcs in G, instead of just 1 chain for single machines.

Setup times: Certain machines must be set up before they can process the next operation. This happens when tools must be switched, when the machine must be cleaned, and so on. During this setup the machine cannot process operations. This is modelled by giving the arcs representing the schedule weights that are equal to the required setup times. Also, an algorithm is needed that schedules the operations taking the setup times into account. In [15] an algorithm is given that solves single-machine scheduling problems with setup times.

Multiple resources: Often, an operation needs more than one resource simultaneously during processing. For example, an operation needs a scarce pallet on which it must be fixed, a certain (unique) tool, or an operator that tends the machine. We model this by adding disjunctive edges to G that connect all operations that need the same resource. The edges need to be oriented such that they represent the schedules on the different resources. We use two approaches to deal with multi-resources aspects:

1. In the integral approach, we see every resource group as a machine group that needs to be scheduled. Every resource group becomes a bottleneck group in the SB procedure. This approach is useful when the number of additional resource groups is limited.
2. The hierarchical approach is typically used to model *Flexible Manufacturing Cells (FMCs)*. Usually, an FMC consists of a parallel machine group and a large set of unique tools that can only be used by the machines of the FMC. Due to the large number of unique tools, it is better to schedule the FMC taking into account the tool restrictions, rather than scheduling each tool separately, because this would be very time consuming. Meester and Zijm [13] present a hierarchical algorithm to schedule an FMC. This is why we call it the hierarchical approach.

Down times: The machines in the shop may have different availability times: some machines work 24 hours a day, other machines only work 8 hours a day. Also, machines might be unavaible due to maintenance. We distinguish two types of down times: *preemptive* and *non-preemptive* down times. We call a down time preemptive, when it is allowed that an operation starts before this down time and finishes after it. For example, a weekend is a preemptive down time: an operation may start at Friday afternoon and finish at Monday morning. We model preemptive down times by increasing the weight of a node with the length of the down time if the corresponding operation starts before and finishes after the down time. We also need an algorithm that takes the down times into account. Carlier's algorithm (see [4]) can easily be extended to deal with preemptive down times.

When each operations needs to be processed completely before or completely after a down time, this down time is called a non-preemptive down time. Maintenance, for example, is usually a non-preemptive down time: during maintenance, no operation may be clamped on the machine. The non-preemptive down times are modelled as operations that need to be processed in a certain interval. An algorithm to solve this problem can be found in Westra [16].

Convergent and divergent job routings: In the classic job shop problem, each job is a *chain* of operations. In practice, however, the job routings can be convergent or divergent. A convergent job routing occurs when some components are assembled to another component or to a final product. An example of a divergent job routing is the cutting of a metal plate. Before the cutting the plate needs some operations such as surface treatments. After the cutting, the different parts follow their own routing through the shop. We model this by allowing that G contains more than one ingoing or outgoing job arc.

Open shops: In the classic job shop, the sequence in which the operations of a job must be processed is given. In open shop problems, this sequence is not given. A constraint is that the operations of one job cannot be processed simultaneously, but the sequence in which the operations of a job are processed is free. We model this by introducing for each job a single machine on which the operations of this job must be processed. The operations of this job therefore need at least two resources: the newly introduced machine and the machine group on which the actual processing takes place.

4 Tests

The SB procedure has proven to be an effective algorithm for the classic job shop problem, cf. Lawler et al. [11]. In this section, we test how the procedure performs when used in an assembly shop. We study the influence of static and dynamic scheduling, setup times, batch sizes, and the job arrival process. Meester [12] tests the performance of the SB procedure in a shop with multi-resource aspects.

4.1 Test shop

We consider a shop with 4 machines $M_1, \ldots, M_4$. M_1 produces component A, M_2 produces component B, and M_3 produces component C. A job is an order to produce a component or an order to produce a product consisting of different components that need to be assembled. M_4 is used to assemble components. All possible combinations of components are considered. So, jobs arrive for producing A, B, C, AB, AC, BC, and ABC. Each product has an equal chance of being chosen. Jobs arrive at the shop according to a Poisson process with a mean interarrival time of 70 time units. We draw the processing times of an operation on M_1, M_2, and M_3 from a discrete uniform distribution on the interval $[85, 115]$. We consider problems with and without setup times on M_4. If no setup times occur, we draw

		to			
		AB	AC	BC	ABC
	$-$	200	200	200	300
	AB	0	120	120	100
from	AC	120	0	120	100
	BC	120	120	0	100
	ABC	20	20	20	0

Table 2: Setup times on M_4.

the processing times for operations on M_4 from a discrete uniform distribution on the interval $[90, 130]$. Otherwise, we draw on the interval $[65, 85]$. Table 2 gives the setup times for M_4 in the latter case. The logic behind these setup times is the following. If a component is assembled, then a specific tool is needed. If a component is not assembled, this tool may not be on the machine. Mounting a tool to the machine takes 100 time units; unmounting a tool takes 20 time units. The due date of job J_j is set to $d_j = r_j + P_j + D_j \cdot NO_j$, with P_j the total processing time of J_j, and NO_j the number of operations of J_j. We draw D_j from a discrete uniform distribution on the interval $[200, 400]$.

One of the influences we want to study is the influence of static and dynamic scheduling. Static scheduling means that all jobs of an instance are scheduled at once. Dynamic scheduling means that we split a set of jobs into smaller sets. In the tests, we do this in the following way. First, let t be such that on the average 200 jobs arrive at the shop during the interval $[0, t]$. In this case, $t = 200 \cdot 70 = 14,000$. Schedule all jobs that arrive in the interval $[0, 2 \cdot t]$. We call this the first run. All operations that start in the interval $[0, t]$ are fixed. Now, all operations that were scheduled in the first run but were not fixed, and all jobs that arrive in the interval $[2 \cdot t, 3 \cdot t]$ are scheduled. This is the second run. All operations that start in the interval $[t, 2 \cdot t]$ are fixed. This process continues until all operations are fixed. For static scheduling, we generate instances with 500 jobs. For dynamic scheduling we generate instances with 2,000 jobs.

4.2 Priority rules

In practice, the most common way to schedule jobs in a shop is by use of priority rules. In the literature, priority rules are extensively tested for assembly shops, cf. Russell and Taylor [14] and Fry et al. [10]. We compare the performance of the SB procedure with two types of priority rules which have been shown to perform well for assembly shops without setup times:

1. $\rho_{ij} = -d_j$, and
2. $\rho_{ij} = -slack_{ij}$,

S/ NS	stat/ dyn	time	
		pri	SB
NS	stat	1.0	4.8
NS	dyn	4.0	29.0
S	stat	49.1	16.3
S	dyn	173.3	102.9

Table 3: Mean computation times in seconds.

with ρ_{ij} the priority of operation O_{ij}, and $slack_{ij}$ the slack of this operation. So, the first priority rules focuses on the due date of the job; the second focuses on the slack that operations have. For operations on a machine with setup times, we decrease the priority of an operation with $\beta \cdot setup$, with $setup$ the required setup time. We vary β from 0 to 20 with steps of 0.5. So, for problems without setup times, we evaluate 2 priority rules; for problems with setup times, we evaluate 82 priority rules.

4.3 Implementation of the SB procedure

Several authors propose improvements of the SB procedure. Dauzere-Peres and Lasserre [7] make a very important observation. They show that the operations on a machine are in general dependent, although they are treated as being independent in the original SB procedure. See, for example, Figure 3. Operations O_{11} and O_{22} are scheduled on a bottleneck machine. If we want to schedule the non-bottleneck machine on which operations O_{12} and O_{21} need to be processed, then O_{12} must be scheduled before O_{21}. Moreover, after the completion of O_{12}, O_{22} and O_{11} must be processed before O_{21} can be processed. So, there is a finish-start relation between O_{12} and O_{21}; this relation is called a *delayed precedence constraint*.

We choose to use the delayed precedence constraints, because Dauzerre-Peres and Lasserre get better results when using them. Balas et al. [2] present an algorithm for solving single-machine problems with precedence constraints optimally. Due to the large instances we generate, we choose to use heuristics to solve the single-machine problems. For the machines without setup times, we use an adapted Schrage algorithm (see Schrage [**?**]). For machines with setup times, we use priority rules. The priority of operation O_{ij} is $\rho_{ij} = late - \beta \cdot NP$, with $late$ the lateness of this operation if we schedule it now, and NP the non-productive machine time. The non-productive machine time consists of setup time and idle time. Again, we vary β from 0 to 20 with steps of 0.5.

4.4 Results

Table 3 gives information about the computation time. Each row in this table gives means of 30 instances. The first column indicates whether setup times occur (S) or not

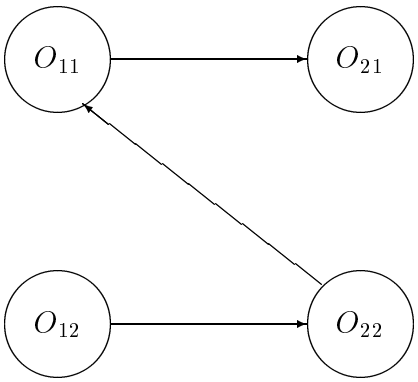

Figure 3: Example of dependent operations.

S/ NS	stat/ dyn	$L_{\max}$		# late jobs		mean tardiness	
		pri	SB	pri	SB	pri	SB
NS	stat	361.0	277.0	32.0	35.2	155.8	123.6
NS	dyn	1159.3	986.1	224.2	254.7	383.6	313.3
S	stat	604.1	354.8	46.4	51.0	195.3	134.8
S	dyn	1039.9	713.0	241.0	263.0	283.1	211.7

Table 4: Performance measures.

(NS). The second column indicates whether the instances are scheduled statically (stat) or dynamically (dyn). The instances that are scheduled statically have 500 jobs; instances that are scheduled dynamically have 2,000 jobs. The third column gives the mean computation time in seconds on a HP workstation to evaluate all priority rules. For instances without setup times, we evaluate 2 priority rules; for instances with setup times, we evaluate 82 priority rules. The last column gives the mean computation time the SB procedure needs. For instances that are scheduled dynamically, this time is the aggregate time for all the sub-runs. The computation times are very acceptable. Even for problems with 2,000 jobs, the mean computation time does not exceed 3 minutes for the priority rules and does not exceed 2 minutes for the SB procedure.

Table 4 gives some performance measures of the schedules of the different approaches. Each row gives again means of 30 instances. The first two columns indicate again whether setup times occur and whether the SB procedure solves the instances statically or dynamicaly. The third and the fourth column give information about the maximum lateness. The third column gives the best value of the maximum lateness of all schedules generated by the priority rules. The fourth column gives the value of the maximum lateness of the schedule generated with the SB procedure. We see that the SB procedure performance considerably better than all priority rules. Columns five and six give information about the second performance measure we consider: the number of late jobs, i.e., the number of jobs that are completed after their due date. For this measure, the best priority rule gives in the mean better results than the SB procedure. The last criterium we consider is mean tardiness for which columns seven and eight give information. In this paper, the mean tardiness is defined on the late jobs: given that a job is too late, how much is it in the mean too late. For this criterium, the SB procedure is better again. We recognize the primary objective function of the SB procedure: minimizing the maximum lateness. The SB procedure prefers solutions with more jobs less late over solutions with few jobs much too late.

What we also see in Table 4 is that the performance of the SB procedure remains good if we use it dynamically. This is very useful if we want to use it in practice, because those problems might be too large to schedule statically.

Table 4 compares the SB procedure for different performance measures with the best priority rule for each measure. The best priority rule for, e.g., the maximum lateness may

be another rule than the one that gives the best solution for the mean tardiness criterium. In practice, however, one is interested in a solution that has a good maximum lateness, that has few jobs too late, and that has a small mean tardiness. Table 5 compares the solution values of the SB procedure with the solution values of the priority rule that gives the best maximum lateness. Columns three and six give again the solution value of the best priority

S/	stat	# late jobs			mean tardiness		
NS	dyn	pri		SB	pri		SB
		best	ml		best	ml	
NS	stat	32.0	32.2	35.2	155.8	157.9	123.6
NS	dyn	224.2	225.4	254.7	383.6	388.3	313.3
S	stat	46.5	59.5	51.0	195.3	221.1	134.8
S	dyn	241.0	280.7	263.0	283.1	305.6	211.7

Table 5: Comparison of solution values.

rule. The columns that have the label 'ml' give the values of the performance measures of the schedule generated with the priority rule that gives the best maximum lateness. The mean tardiness of the best priority rule is worse than the value of the schedule generated by the SB procedure. Of course, the values deteriorate in the column 'ml'. The best priority rule gives better results for the number of late jobs than the SB procedure. For the problems without setup times, the values in the column 'ml' deteriorate, but they are still better than the value of the SB schedule. For the problems with setup times, however, the values in the column 'ml' are worse than the values of the SB schedule. We conclude that compared with priority rules, the SB procedure produces a schedule which is very good. It has a good maximum lateness and mean tardiness. The number of late jobs in this schedule is acceptable.

In the tests above, a job is always an order to produce *one* component or product. An interesting influence to study is the impact of batch arrivals, that is, a job is an order to produce a number of the same components or products. In the remaining test, we determine a batch size of the jobs that arrive at the shop. We draw the batch size of a job from a discrete uniform distribution on the interval $[1, 5]$. The mean batch size is therefore 3. In Table 6, we give test results for the case that the arrival time of the next job depends on the batch size of this job. More specifically, assume that q is the batch size of this job. Then, the interarrival time for the next job is drawn from a negative exponential distribution with a mean of $100 \cdot q$ time units. With this arrival process, we try to simulate the situation that the sales department of a company tries to spare the production department after selling a large order. Note that we increased the mean processing time with a factor 3, whereas we increased the mean interarrival time with a factor $\frac{100}{70} \cdot 3 > 4$. We see from Table 6 that the due date performance of the shop deteriorates, though we decreased the loading. An explanation is found in studies in queueing theory, that say that the mean waiting time of the jobs increases with increasing variation in the arrival process, cf. Whitt [17]. The SB

S/ NS	stat/ dyn	$L_{\max}$		# late jobs		mean tardiness	
		pri	SB	pri	SB	pri	SB
NS	stat	1005.5	906.8	57.6	66.5	251.6	246.6
NS	dyn	1767.7	1644.3	244.6	274.7	307.6	302.1
S	stat	872.6	796.1	49.1	58.0	218.6	215.3
S	dyn	1368.8	1268.8	209.5	240.5	243.8	245.4

Table 6: Results for batch arrivals.

procedure still works comparable to the situation with no batch arrivals. The differences, however, are smaller. We feel that this is explained by the smaller load of the shop.

Due to the large variation in the arrival process, the due date performance of the shop in the previous test was bad. Now, we study the due date performance of the shop with a more regular arrival process. We use an Erlang-4 distribution for the interarrival times, which is the summation of 4 drawings from a negative exponential distribution. The variance of an Erlang-4 distribution is half the variation of a negative exponential distribution with the same mean. The mean interarrival time is again 300 time units. Table 7 gives test results for this situation. We see that indeed the due date performance of the shop improves a lot

S/ NS	stat/ dyn	$L_{\max}$		# late jobs		mean tardiness	
		pri	SB	pri	SB	pri	SB
NS	stat	422.4	404.6	25.1	25.9	124.0	120.6
NS	dyn	728.7	689.7	102.2	107.7	135.8	131.0
S	stat	364.8	348.4	21.9	23.0	115.0	110.8
S	dyn	562.3	521.1	88.5	92.3	113.4	111.2

Table 7: Test results with Erlang-4 arrival process.

with a more regular arrival process. For a company it might be very advantageous to try to regulate the arrival process. Again, the performance of the SB procedure in this test is comparable to its performance in the previous tests.

5 Conclusions

We presented the SB bottleneck procedure for the classic job shop problem. We showed that this procedure can easily be adapted to deal with practical situations such as transportation times, multiple resources, and down times. We compared the performance of this approach with priority rules which are most common in practice. Setup times occur in the shop we considered, and the jobs had convergent routings. The SB procedure performs well in

comparison with the priority rules. For part manufacturing shops, due date performance is very important. Therefore, it might be a good choice to replace the planning of a shop with priority rules by a more sophisticated approach like the SB procedure. Also, attention should be paid to the regulation of the arrival process.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[2] E. Balas, J.K. Lenstra, and A. Vazacopoulos. The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41:94–109, 1995.

[3] J.D. Blackburn. *Time-Based Competition, The Next Battle Ground in American Manufacturing*. Richard D. Irwin, Homewood, Ill., 1991.

[4] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[5] J. Carlier. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research*, 29:298–306, 1987.

[6] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35:164–176, 1989.

[7] S. Dauzere-Peres and J.-B. Lasserre. A modified shifting bottleneck procedure for job-shop scheduling. *International Journal of Production Research*, 31:923–932, 1993.

[8] W.E. Deming. *Quality, Productivity, and Competitive Position*. MIT Center for Advanced Engineering Study, Cambridge, Mass., 1982.

[9] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J.F. Muth and G.L. Thompson, editors, *Industrial Scheduling*, pages 225–241. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[10] T.D. Fry, M.D. Oliff, E.D. Minor, and G.K. Leong. The effects of product structure and sequencing rule on assembly shop performance. *International Journal of Production Research*, 27(4):671–686, 1989.

[11] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. Technical report, Centre for Mathematics and Computer Science, Amsterdam, 1989.

[12] G.J. Meester. *Multi Resource Scheduling ??????* PhD thesis, University of Twente, Department of Mechanical Engineering, Laboratory of Production and Operations Management, 1995.

[13] G.J. Meester and W.H.M. Zijm. Multi-resource scheduling for an FMC in discrete parts manufacturing. In M.M. Ahmad and W.G. Sullivan, editors, *Flexible Automation and Integrated Manufacturing*, pages 360–370. CRC Press Inc., Atlanta, 1993.

[14] R.S. Russell and B.W. Taylor III. An evaluation of sequencing rules for an assembly shop. *Decision Sciences*, 16:196–212, 1985.

[15] J.M.J. Schutten, S.L. van de Velde, and W.H.M. Zijm. Single-machine scheduling with release dates, due dates and family setup times. Technical Report LPOM-93-4, University of Twente, Department of Mechanical Engineering, 1993.

[16] S.J. Westra. Het een-machine scheduling probleem met gefixeerde jobs. Master's thesis, University of Twente, Department of Mechanical Engineering, Laboratory of Productions and Operations Management, 1994.

[17] W. Whitt. Approximations for the $gi|g|m$ queue. *Production and Operations Management*, 2:114–161, 1993.