Practical job shop scheduling

Article *in* Annals of Operations Research · October 1998

DOI: 10.1023/A:1018955929512

1 author:

Marco Schutten
University of Twente
63 PUBLICATIONS **1,021** CITATIONS

SEE PROFILE

Practical job shop scheduling

J.M.J. Schutten
Faculty of Mechanical Engineering,
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: m.schutten@wb.utwente.nl

The Shifting Bottleneck procedure is an intuitive and reasonably good approximation algorithm for the notoriously difficult classical job shop scheduling problem. The principle of decomposing a classical job shop problem into a series of single-machine problems can also easily be applied to job shop problems with practical features, such as transportation times, simultaneous resource requirements, setup times, and many minor but important other characteristics. We report on the continuous research in the area of extending the Shifting Bottleneck procedure to deal with those practical features. We call job shops with such additional features *practical* job shops. We discuss experiences with the Shifting Bottleneck procedure in a number of practical cases.

1. Introduction

Manufacturing processes have become increasingly complex, and so have their planning and control. The market requires high product variety, high quality, short leadtimes, and an accurate delivery performance. Timely delivery is now a major competitive edge; cf. Blackburn [5]. The market requirements force manufacturing companies to be flexible: They need to engineer and produce a large variety of products efficiently with short leadtimes and reliable due dates. Technological progress and information technology made this, in principle, possible. Setup time reduction, for example, enables companies to produce in smaller batches and therefore to be more flexible.

Technological progress, however, has led to more complex planning and control problems. In the past, each machine had its own operator, its own tools, and so on. Nowadays, tools are more versatile, hence can be used by various different machining centres. These tools are, however, at the same time more expensive. Therefore, a company often decides to cut its tool investments. Similarly, operators are expensive, but often they only need to tend a machine during part of the operations, e.g., to position a job on the machine. This enables the operators to tend more than one machine; therefore, the number of operators is often smaller than the number of machines. Such an operator and tool sharing clearly increases the interdependency of the machines, because a unique tool can only be used by one machine at a time and an operator can

only tend one machine at a time. The planning and control must take these dependencies into account and the machines must be planned and controlled simultaneously to realize short leadtimes and a good delivery performance.

In the operations research literature, machine scheduling problems, such as the job shop problem, are a popular area of research and a high level of algorithmic design and analysis has been achieved. These problems are, however, "clean" in that they ignore the nasty side constraints that often occur in practice. In contrast, the production literature addresses such practical scheduling problems, but the emphasis is mainly on problem formulation and giving practical solutions. The algorithms that are proposed are usually myopic: Often, a number of priority rules is tested and the best one is proposed.

This paper tries to fill the gap between the operations research literature and the production literature by extending the Shifting Bottleneck (SB) procedure of Adams et al. [1] for the classical job shop to deal with practical features, such as transportation times and convergent job routings. These practical features usually prohibit a good theoretical analysis of the problem. The SB procedure decomposes the problem of scheduling a classical job shop into a series of single-machine scheduling subproblems. For practical applications, computation time is very important. In general, the SB procedure produces good solutions for job shop problems in relatively short computation time; cf. Vaessens et al. [21]. This is why we use the SB procedure, instead of randomized local search methods like tabu search and simulated annealing that take more computation time.

The original paper by Adams et al. has prompted quite some research on the SB procedure, which has taken two directions. The first concerns algorithmic improvements of the procedure; see, for instance, Dauzère-Pérès and Lasserre [11], Balas et al. [2], and Balas and Vazacopoulos [3]. The second direction concerns the adjustment of the SB procedure to more *practical* job shops, i.e., job shops with practical side features such as simultaneous resource requirements and setup times. Ovacik and Uzsoy [16] use an adapted SB procedure for scheduling semiconductor testing facilities. Their computational experiments with real-life data show that this procedure significantly outperforms dispatching rules both in terms of solution quality and robustness. Ivens and Lambrecht [13] discuss some practical extensions of the SB procedure, such as release and due dates, setup times, and convergent job routings.

In this paper, we report on the research in the area of extending the SB procedure with practical features. This paper is partly based on the work of Meester [14], who focuses on simultaneous resource requirements, and on the work of Schutten [19] who focuses on problems with setup times and convergent job routings. Also, part of this paper is based on graduation projects at the University of Twente. The main goal is always to deliver the jobs in time, i.e., before their due date. The SB procedure with extensions is part of a commercial shop floor control system called JOBPLANNER.

The plan of this paper is as follows. In the next section, we discuss the classical job shop problem. In section 3, the SB procedure will be discussed, while possible

extensions of this procedure are outlined in section 4. In section 5, we report on some algorithmic improvements of the SB procedure. Finally, in section 6, we end with some conclusions.

2. Classical job shop problem

The problem of scheduling jobs in a machine shop is often modeled as a *classical* job shop problem, which is 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 $J_1, J_2,\ldots, J_n$ needs 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 $O_{1j}, O_{2j},\ldots, O_{n_j, j}$, where n_j denotes 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. O_{ij} needs uninterrupted processing on machine μ_{ij} during a given non-negative time p_{ij}. The objective is usually to find a schedule that minimizes the *makespan*, that is, to find a schedule in which the time to process all jobs is minimal.

The classical job shop problem is one of the hardest combinatorial optimization problems. For example, a problem with only 10 jobs and 10 machines, proposed by Fisher and Thompson [12], remained unsolved for more than 25 years, in spite of the research effort spent on it. Due to its intractibility, several authors develop branch and bound algorithms to solve the problem; cf. Carlier and Pinson [9] and Brucker et al. [6]. Most algorithms for the classical job shop problem of minimizing makespan use a disjunctive graph G to represent an instance. For each operation O_{ij}, G has a node υ_{ij} with weight p_{ij}. G also has two auxiliary nodes s and t, both with weight 0. For each pair of consecutive operations O_{ij} and $O_{i+1, j}$, G has a conjunctive arc $(\upsilon_{ij}, \upsilon_{i+1, j})$ ($i = 1,\ldots, n_j - 1$). Also, there are arcs from s to υ_{1j} and arcs from $\upsilon_{n_j, j}$ to t ($j = 1,\ldots, n$). Between every pair of operations O_{ij} and O_{kl} that must be processed on the same machine, G has a disjunctive edge. The operations that are connected by disjunctive edges form a *conflict set*: They cannot be processed simultaneously. The weights of all arcs and edges are 0. The crux is now that each orientation of all disjunctive edges such that G contains no directed cycle results in a feasible schedule. The makespan of this schedule equals the length of the longest weighted path from s to t in the resulting graph. Accordingly, minimizing makespan comes down to finding an orientation of the edges such that the length of the longest weighted path from s to t is minimal.

Table 2 shows the data of an instance with 3 machines and 3 jobs, where each job consists of 3 operations. Figure 1 shows the graph representing this instance. Figure 2 gives a representation of the 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 $(\upsilon_{11}, \upsilon_{33})$. Note that in the resulting graph each node υ_{ij} has indegree and outdegree of at most 2. The longest weighted path from s to t is the path $s - \upsilon_{11} - \upsilon_{21} - \upsilon_{32} - t$ with length 19.

Table 1

Data for example instance.

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

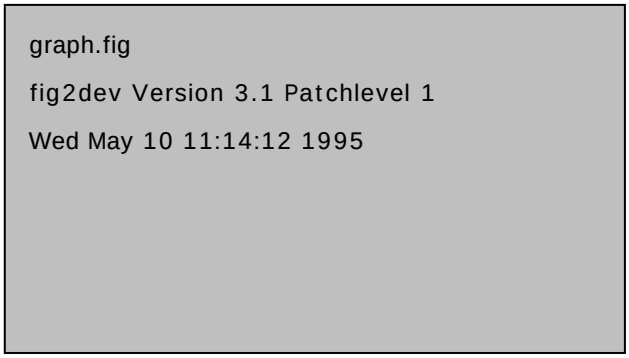

Figure 1. Graph representing example instance.

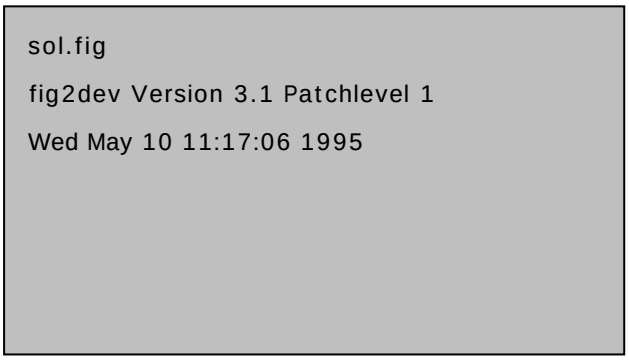

Figure 2. Graph representing a feasible solution.

3. Shifting Bottleneck procedure

Due to its intractability, various approximation algorithms have been proposed for the job shop problem, including tabu search, simulated annealing, and genetic algorithms. We refer to Vaessens et al. [21] for a computational study of the performance of the most prominent. One of them is the Shifting Bottleneck (SB) procedure of

Adams et al. [1]. The SB procedure is an intuitive algorithm that decomposes the job shop problem into a series of single-machine subproblems. It schedules the machines one by one and focuses on bottleneck machines. The method heavily relies on computation of longest weighted paths in the graph G discussed above:

- the longest weighted path from node s to node υ_{ij}, not including the weight of υ_{ij}, defines the earliest possible starting time of operation O_{ij}, that is, it defines a *release date* r_{ij} for operation O_{ij};
- the length of the longest weighted path from node υ_{ij} to node t, not including the weight of υ_{ij}, equals the minimum time the shop needs to process all jobs after the completion of operation O_{ij}, that is, it defines a *run-out time* q_{ij} for operation O_{ij};
- if all machines are scheduled, then the longest weighted path from s to t equals the makespan of this schedule.

The procedure starts by removing all disjunctive edges from G, labeling all machines as non-bottleneck machines, and computing the longest paths. The longest path computations take $O(N)$ time, where $N = \sum_{j=1}^{n} n_j$, since each node υ_{ij} has out-degree at most two. Next, all machines are scheduled separately. We then need to solve m single-machine scheduling problems of minimizing makespan where the operations have release dates and run-out times. If we give operation O_{ij} a due date $d_{ij} = -q_{ij}$, then these problems are equivalent to the single-machine scheduling problems of minimizing *maximum lateness* $L_{\max}$, which measures the maximum difference between the completion times and the due dates of the operations. The release dates and run-out times of the operations follow from the longest path computations. Adams et al. use Carlier's [7] optimization algorithm to solve these single-machine scheduling problems. The machine with the largest resulting makespan is labeled as a bottleneck machine. The schedule of this machine is fixed by adding to G the arcs representing the sequence on this machine. Now, longest paths are recomputed and the non-bottleneck machines are scheduled subject to the updated release dates and run-out times. The machine with the largest resulting makespan is also labeled as a bottleneck machine. The bottleneck machines are now *rescheduled* in a special bottleneck optimization step. G is changed, such that the machine arcs represent the, possibly changed, sequences on the bottleneck machines. Longest paths are again computed, the non-bottleneck machines are scheduled separately, and so on. This process continues until all machines have been labeled as bottleneck machines.

4. Extensions of the SB procedure

The SB procedure consists of some generic steps, such as the computation of longest paths in G. Apart from the condition that G may not contain directed cycles, we imposed no properties on G. The decomposition of a job shop problem results for

the classical job shops in single-machine scheduling subproblems. In this section, we show that by changing the properties of G, by redefining the conflict sets, and by changing the algorithms for the machine scheduling subproblems, we can deal with various extensions of the classical job shop problem.

Practical instances may be very large and the machine scheduling subproblems can then often not be solved to optimality in reasonable time. In the SB procedure, it is possible to use a heuristic for the machine scheduling subproblems. Below, we discuss possible extensions of the classical job shop problem and how we model them. Although all possible combinations of those extensions are allowed, we only consider one extension at a time. We stress that the decomposition principle proceeds along the lines Adams et al. proposed; we only adjust G, specify the resulting conflict sets, and characterize the resulting machine scheduling subproblems.

4.1. Release and due dates

In the classical job shop problem, all jobs become available for processing at the same time. This is seldom true in practice, where jobs may have different release dates. Suppose now that job J_j has release date r_j. If we give the arc (s, υ_{1j}) weight r_j, then the longest weighted path from s to υ_{1j} is at least r_j. Thus, we ensure that $r_{1j} \geq r_j$ and that operation O_{1j} does not start before time r_j. Also, it is possible that we cannot process operation O_{ij} before some point in time t_{ij} $(i = 1,\ldots,n_j)$ because some tool or material is not available before that time, with $t_{ij} > r_j + \sum_{k=1}^{i-1} p_{kj}$. We model this by adding an arc from s to υ_{ij} with weight t_{ij}.

In the classical job shop problem, the objective is usually to minimize makespan. In practice, jobs for different customers have different due dates. It is then appropriate to have an objective function that measures the due date performance. Let d_j denote the due date of job J_j. Consider now the objective of minimizing *maximum lateness* $L_{\max}$ with $L_{\max} = \max_{j=1,\ldots,n}\{C_j - d_j\}$ and C_j the completion time of J_j. The objective minimizing $L_{\max}$ is represented in G by giving the arcs $(\upsilon_{n_j,j}, t)$ weight $-d_j$ $(j = 1,\ldots,n)$. Note that this objective generalizes minimizing makespan.

Thus, to deal with job release and due dates, we do not change the considered conflict sets and the machine scheduling subproblems. We only change the weights of some arcs in G. These changes are made *off-line*, i.e., before the SB procedure starts. If also the operations have release dates, then we must add some arcs to G (off-line).

4.2. Setup times

A machine may have to be set up before it can process the next operation. This happens, for instance, when tools must be switched off-line and when the machine must be cleaned between two operations. During a setup, the machine cannot process any operation.

Suppose that a partial schedule on the machine with setup times is $O_{gh} - O_{ij}$. The setup time between O_{gh} and O_{ij} is $s_{gh,ij}$. In G, we model this setup time by giving

weight $s_{gh,ij}$ to the machine arc $(\upsilon_{gh}, \upsilon_{ij})$. The longest weighted path from υ_{gh} to υ_{ij} in G is then at least $s_{gh,ij}$. This ensures that there are at least $s_{gh,ij}$ time units between the completion of O_{gh} and the start of O_{ij}, which leaves room for the needed setup. Note that the incorporation of setup times requires *on-line* changes of G, i.e., during the execution of the SB procedure. As soon as we know that O_{gh} and O_{ij} are processed consecutively, we add to G the machine arc $(\upsilon_{gh}, \upsilon_{ij})$ with weight $s_{gh,ij}$.

In the standard SB procedure, we need to solve single-machine problems of minimizing makespan where the jobs have release dates and run-out times. Now, the needed setup times between the execution of the operations need to be taken into account. For single-machine scheduling problems with family setup times, Schutten et al. [20] present a branch and bound algorithm that solves instances with up to 40 jobs to optimality.

Accordingly, the SB procedure can deal with setup times by changing the weights of the machine arcs in G (on-line), and by having an algorithm for the machine scheduling subproblems with setup times. The conflict sets do not change.

In the Sheet Metal Factory of DAF trucks in Eindhoven (The Netherlands), setup times occur when changing the moulds of the presses. Belderok [4] tests the SB procedure with setup times in this factory. His computational experiments indicate that a significant leadtime reduction and a better due date performance is possible, in particular for the Sheet Metal Press department. For example, Belderok tests the procedure on a real-life set of jobs that were processed in five days. The makespan of the schedule generated by the SB procedure is less than three and a half days.

4.3. Parallel machines

In the classical job shop, every operation requires a specific machine. In practice, an operation may sometimes be performed by any machine from a group of parallel machines. Parallel machine scheduling reduces to assigning each operation to one of the machines and sequencing the operations assigned to the same machine.

The decomposition of the job shop scheduling problem results in this case in a series of parallel-machine scheduling problems where the jobs have release dates and run-out times. If the machines in a group are identical, then we may use Carlier's [8] algorithm to solve these machine scheduling subproblems.

If the machines in the group are not identical, that is, if the processing time of an operation differs across machines, then the weight of the corresponding node changes during the execution of the SB procedure. If a parallel-machine group is labeled as a bottleneck, then the weight of the corresponding node is equal to the processing time on the machine to which the operation is assigned. Otherwise, the weight is equal to the smallest processing time of this operation.

In G, we have for each machine in this group a chain of arcs representing the sequence on this machine. In figure 3, the bold arcs represent the sequences for a group of two parallel machines with $O_{12}-O_{21}$ the sequence on the first machine and

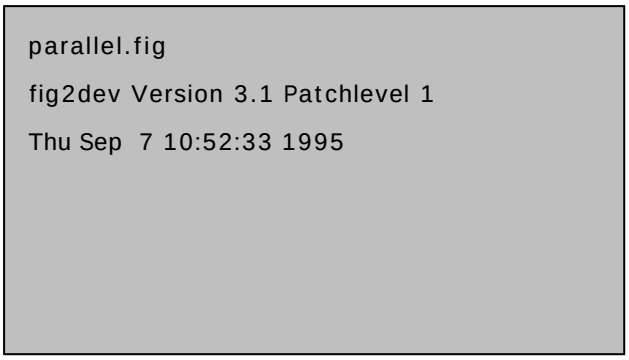

Figure 3. Representation of a parallel machine schedule.

$O_{14} - O_{23} - O_{25}$ the sequence on the second machine. Note that we do not specify a sequence for all operations in the conflict set resulting from the parallel-machine group. In figure 3, e.g., we do not specify a sequence between operations O_{12} and O_{14} because they are processed by different machines in the group.

Hence, the incorporation of parallel-machine groups requires an algorithm to solve the resulting parallel-machine scheduling subproblems. If the processing times are machine dependent, then this requires on-line changing of node weights.

4.4. Transportation times

In practice, it may be impossible to start operation O_{ij} immediately after the completion of operation $O_{i-1,j}$ because the product must first be transported from machine $\mu_{i-1,j}$ to machine μ_{ij}. If the transportation of a product always starts immediately after the completion of the operation, then we model this by giving arc $(\upsilon_{i-1,j}, \upsilon_{ij})$ a weight that is equal to the transportation time. This creates enough time between the completion of $O_{i-1,j}$ and the start of O_{ij} to transport the product to the next machine. Note that we only change G off-line to deal with this type of transportation time.

Reesink [17] tests the SB procedure with transportation times at Stork Plastics Machinery in Hengelo (The Netherlands). He also uses transportation times to model operations that are subcontracted. These operations are presumed to have a fixed leadtime. On randomly generated test sets, Reesink tests the influence of transportation times of different magnitude on the performance of the SB procedure. In his tests, there are transportation times between any two consecutive operations. His tests indicate that the increase of the leadtimes of the jobs is less than the total increase of the transportation times. Belderok [4] uses transportation times to make the resulting schedule more robust, that is, a small deviation in the processing time of an operation does not necessarily result in a need for rescheduling.

If transportation capacity is limited, then transportation is an operation that we need to schedule as well; it is then uncertain when transportation takes place and, accordingly, when the next operation can start. We model, for example, transportation with vehicles that can only transport one job at a time as a parallel-machine group with setup times. Each vehicle is seen as a machine in this group, with the number of machines being equal to the number of transportation vehicles. Figure 4 shows the

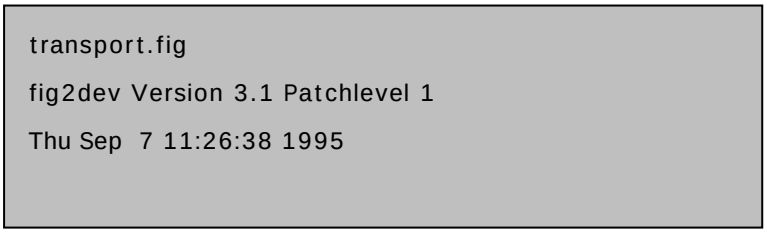

Figure 4. Schedule of transportation vehicle.

representation of an instance with two transportation operations. The nodes corresponding to these transportation operations are indicated by squares instead of circles. The figure represents the schedule $O_{21}-O_{32}$ for the transportation vehicle. This means that the vehicle must pick up job J_1 at machine μ_{11} and transport it to machine μ_{31}. This transportation takes 7 time units; after this, the vehicle must pick up J_2 from machine μ_{22}. Since the vehicle can only transport one job at a time, it travels empty from μ_{31} to μ_{22}. We see this empty travel time for the vehicle as a setup time, and model it accordingly. An open question is the modeling of congestion and blocking of vehicles within the SB procedure.

4.5. Unequal transfer and production batches

A job may be an order to produce a batch of b identical products, not just a single product. An operation O_{ij} of this job is then actually a series of b identical operations: $O_{ij} = (O_{i,1,j}, O_{i,2,j}, \ldots, O_{i,b,j})$. If the b identical products need to be processed *consecutively* on each machine, then O_{ij} is called a *production batch*. We assume that a production batch needs to be processed continuously, i.e., without idle time, on the machines. Suppose now that we may transport $O_{i,k,j}$ ($k = 1,\ldots,b-1$) to the next machine immediately after its completion. If we do this, then it may result in a smaller completion time on the next machine for the production batch. We call $O_{i,k,j}$ a *transfer batch*. For problems with $p_{ij} > p_{i+1,j}$, we shift the batches on the second machine to the right, such that no idle time between the batches on this machine exists; see figure 5 for an example with $b = 4$. Note that the difference in time between the completion of O_{ij} and the start of $O_{i+1,j}$ is at least $(-(b-1)/b) \cdot p_{i+1,j}$ time units. For problems with $p_{ij} \leq p_{i+1,j}$, the transfer batches may immediately be processed on the next

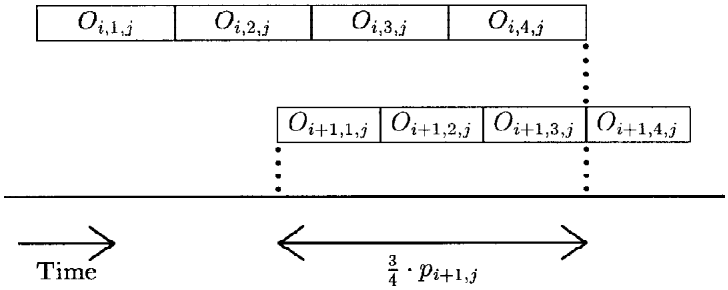

Figure 5. Transfer batches with $p_{ij} > p_{i+1,j}$ and $b = 4$.

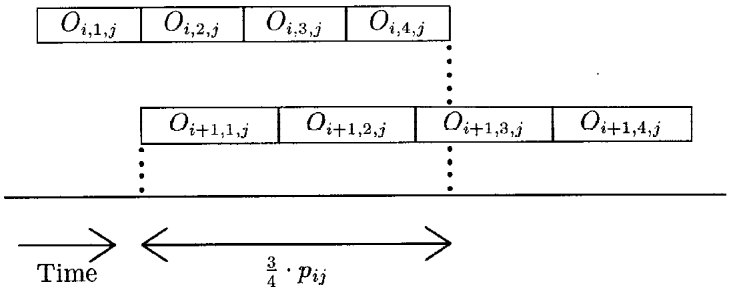

Figure 6. Transfer batches with $p_{ij} \leq p_{i+1,j}$ and $b = 4$.

machine after transporting it; see figure 6. The difference between the completion of O_{ij} and the start of $O_{i+1,j}$ is now at least $(-(b-1)/b) \cdot p_{ij}$ time units. The SB procedure can therefore deal with unequal transfer and production batches if we give arc $(\upsilon_{ij}, \upsilon_{i+1,j})$ weight $(-(b-1)/b) \cdot \min\{p_{ij}, p_{i+1,j}\}$. Therefore, unequal transfer and production batches require only an off-line change of the graph G.

4.6. Multiple resources

Often, an operation needs two or more resources simultaneously for its processing. Besides a machine, an operation may need a pallet on which it must be fixed, certain tools, or an operator at the machine. We model this by adding disjunctive edges to G that connect all operations that need the same resource. In figure 7, operations O_{31}, O_{32}, and O_{33} need, besides the machines, the same additional resource. In the SB procedure, we orient those edges such that they represent the schedules on the additional resources. We distinguish two approaches to deal with multiple resources.

(1) *The centralized approach*. In this approach, we see every resource as a machine that needs to be scheduled. We make no difference between machines and other resources. Consequently, every resource becomes a bottleneck machine in the SB procedure. This approach is useful when the number of additional resources

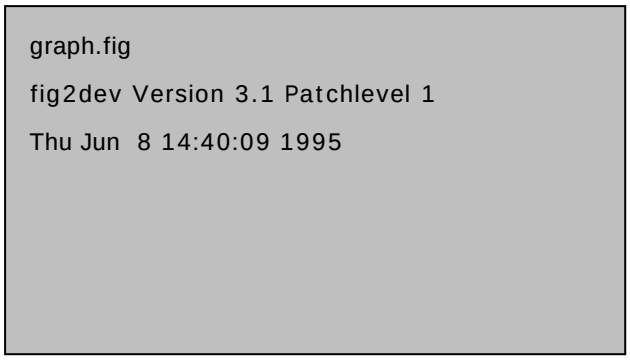

Figure 7. Graph with multi-resource aspects.

is limited. In the centralized approach, we have to define a conflict set for each resource. The resulting machine scheduling subproblems do not change. The interaction of the different resources is handled by the SB procedure; this is why we call this approach the centralized approach.

(2) *The decentralized approach*. If the number of additional resources is large, then the centralized approach may be time-consuming, since we need to label each resource as a bottleneck machine. Sometimes, however, a group of resources may be hardly restrictive. This is, for instance, true for the cutting tools in a *Flexible Manufacturing Cell* (*FMC*). 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. If an FMC is part of the job shop, then in the decentralized approach we need an algorithm to schedule the FMC, that is, we need an algorithm that minimizes the makespan for a group of parallel machines taking into account the tool restrictions, while furthermore the operations have release dates and run-out times. The interaction of the tools and the machines in the FMC is handled by the algorithm for scheduling the FMC, not the SB procedure. This is why we call it the decentralized approach.

Meester and Zijm [15] present a hierarchical algorithm to schedule an FMC. They compare the performance of the algorithm with lower bounds that are found by relaxing the multi-resource constraints. The gap between these lower bounds and the objective values of the generated schedules is sometimes quite large. The authors feel that this is due to the weakness of the lower bounds. In contrast with the modeling of the centralized approach, the modeling of this approach affects both the conflict sets and the machine scheduling subproblems.

Meester [14] tests both approaches on real-life instances. In one case, he tests the centralized approach in the machine shop of Ergon B.V. in Apeldoorn (The Netherlands) where the operations need besides a machine an operator during processing. In another case, Meester tests the decentralized approach in the machine shop of

El-o-Matic B.V. in Hengelo (The Netherlands). This machine shop consists of conventional and computer-controlled machines, among which one FMC with a large number of unique tools. Compared with the planning procedure used by the companies so far, the SB procedure shows in both cases a significant improvement of the due date performance.

Note that the outdegree of the nodes υ_{ij} is not bounded by two any more. This means that the longest path computations in the graph G take $O(N^2)$ time, instead of $O(N)$, because there are now $O(N^2)$ arcs in G. If, however, the outdegree of the nodes υ_{ij} is bounded by k, e.g., the number of additional tools required for each operation is bounded by $k - 2$, then the longest path computations take $O(kN)$ time.

4.7. Down times

The machines in a 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 unavailable due to maintenance. We call a period in which a machine is not available for processing a *down time*. We distinguish two types of down times: *preemptive* and *non-preemptive* down times. We call a down time preemptive if an operation may start before and finish after it. A weekend, for example, is often a preemptive down time: It is often allowed that an operation starts on Friday afternoon and finishes on 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 straddles the down time (an on-line change of G). We also need an algorithm that schedules a machine with preemptive down times. The objective is to minimize makespan and the operations have release dates and run-out times. Reesink [17] shows that Carlier's algorithm (see [7]) can easily be extended to solve this problem. The key idea is to increase the processing time of an operation if it straddles a down time.

If each operation needs to be completely processed either before or after a down time, then this down time is called a non-preemptive down time. Maintenance, for example, is usually a non-preemptive down time: During maintenance, no job may be on the machine. The non-preemptive down times are modeled as operations that need to be processed in a prespecified interval. For each non-preemptive down time, we therefore add a node to G.

If the machines in a shop have non-preemptive down times, then the decomposition of the job shop problem results in machine scheduling subproblems in which the machines have down times. Westra [22] and Woerlee [23] propose algorithms to solve this machine scheduling subproblem. Either algorithm sees a down time as an operation O_{ij} with a release date r_{ij} equal to the start of the down time, a processing time p_{ij} equal to the length of the down time, and a run-out time $q_{ij} = Q - r_{ij} - p_{ij}$, with Q an appropriate constant. Let $C_{\max}(Q)$ denote the optimal value of the resulting problem; Carlier's algorithm (see [7]) can be used to find this value. Clearly, $C_{\max}(Q) \geq r_{ij} + p_{ij} + q_{ij} = Q$ and $C_{\max}(Q)$ is monotonic non-decreasing in Q. Westra and Woerlee

show that if $C_{max}(Q) = Q$, then $C_{max}(Q) \geq C^*_{max}$, with C^*_{max} the optimal solution value of the problem in which each operation associated with a down time must start at its release date. If $C_{max}(Q) > Q$, then $C_{max}(Q) < C^*_{max}$. The problem is then to find the smallest Q such that $C_{max}(Q) = Q$. This can be done by binary search.

Note that both preemptive and non-preemptive down times require on-line changes of G and algorithms to solve machine scheduling problems with down times.

A problem with the modeling of both preemptive and non-preemptive down times is the following: Suppose that M_i is a machine with preemptive down times and that M_i is the first bottleneck machine. Eventually, M_j ($j \neq i$) will also be labeled as a bottleneck machine. The schedule of M_j is then fixed by adding to G the machine arcs representing this schedule, which may delay operations on M_i. It is then possible that an operation that *straddled* a down time can now start only *after* the down time. So, the weight of the corresponding node should no longer be increased with the length of the down time. To resolve this problem, we propose an "intelligent" longest path procedure: When the longest weighted path to a node is computed, the procedure checks the down times of the machines and determines whether the weight of this node should be increased. A similar problem occurs for machines with non-preemptive down times. In this case, our longest path procedure checks whether an operation O_{ij} can be processed entirely before the next down time. If not, the procedure inserts an operation representing the down time just before O_{ij}.

4.8. Convergent and divergent job routings

In the classical job shop problem, each job is a *chain* of operations. In practice, the job routings may be convergent or divergent. A *convergent* job routing occurs when some components are assembled. Figure 8 shows a representation of an instance

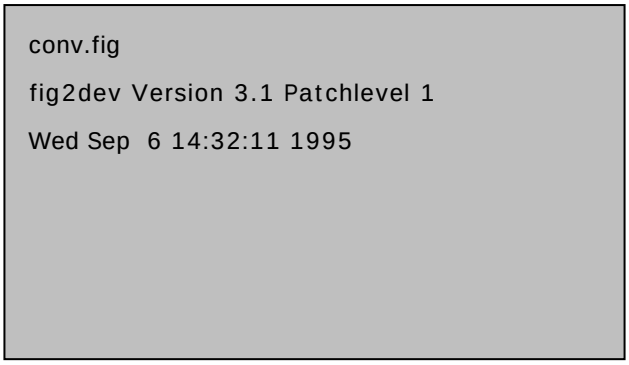

Figure 8. Instance with a convergent job routing.

with a convergent job routing. An example of a *divergent* job routing is the routing of a metal sheet. Before cutting, the sheet needs some operations such as cleaning and surface treatments. After cutting, the different parts of the sheet have their own routings

through the shop. We model this by allowing the nodes in G to have more than one ingoing and outgoing job arc. Note that this modeling of convergent and divergent job routings only influences the properties of G (off-line), not the machine scheduling subproblems and the conflict sets. Schutten [18] tests the performance of the SB procedure for an assembly shop. The influence of setup times and the arrival process is studied in this shop. Also, the influence of dynamic scheduling instead of static scheduling is studied. Test results show that the SB procedure outperforms priority rules on the due date performance indicators maximum lateness and mean tardiness. The priority rules perform better for the performance indicator number of late jobs.

4.9. Open shops

In the classical job shop, the sequence in which the operations of a job must be processed is fixed. In open shop problems, it is not: The operations can be performed in any order, although the operations of the same job cannot be processed simultaneously. We model this by introducing for each job a single artificial machine on which the operations of this job must be processed. Therefore, open shop problems can be seen as special cases of multi-resource problems. The schedule on the artificial machine determines the sequence in which the operations of the corresponding job are processed. Each operation needs at least two resources: The artificial machine and the machine on which the actual processing takes place. Also, we need arcs (s, υ_{ij}) $(j = 1,\ldots, n;\ i = 1,\ldots, n_j)$ to ensure a path from s to every other node. Analogously, we need an arc (υ_{ij}, t) to ensure a path from node υ_{ij} to t. Figure 9 shows the disjunctive graph model of an open shop problem with $n = m = 2$, the data of which are found in table 2. The solid edges in the figure indicate that O_{11} and O_{12} as well as O_{21} and O_{22}

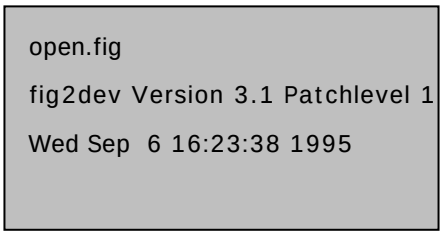

Figure 9. Representation of an open shop problem.

Table 2

Open shop problem.

J_j	μ_{1j}	μ_{2j}	p_{1j}	p_{2j}
J_1	1	2	7	9
J_2	1	2	3	5

need to be processed on the same machine, and can therefore not be processed simultaneously. Likewise, O_{11} and O_{21} as well as O_{12} and O_{22} can not be processed simultaneously because they are operations of the same job. This is indicated by the non-solid edges.

5. Improvements of the SB procedure

Recently, some algorithmic improvements of the SB procedure have been proposed. In this section, we report on three of the improvements that appeared in the literature.

In the SB procedure, operations to be processed on the same machine are treated independently. This is not correct. Figure 10 is the graph that we get after fixing schedule $\upsilon_{13}-\upsilon_{32}-\upsilon_{21}$ on machine M_3 for the instance of which the data can be found in table 1 in section 2. O_{12} and O_{31} need to be processed on machine M_2. If we want

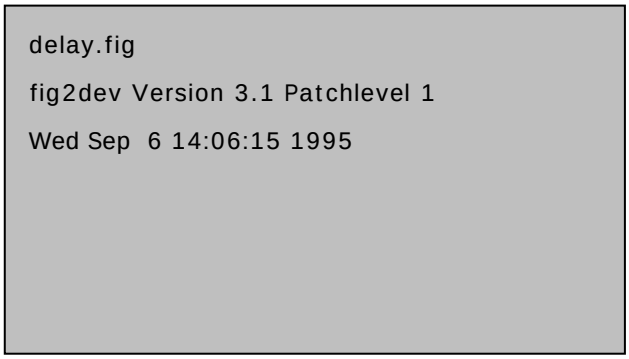

Figure 10. Graph with delayed precedence constraint.

to schedule this machine, then we must schedule O_{12} before O_{31}, because otherwise a directed cycle would occur in the graph and then the schedule would be infeasible. Moreover, after the completion of O_{12}, we must first process O_{22}, O_{32}, and O_{21}, respectively, before we can start the processing of O_{31}. So, there must be a gap of at least $p_{22} + p_{32} + p_{21}$ time units between the completion of O_{12} and the start of O_{31}. A precedence constraint between two operations with the additional constraint that there is a certain delay between these two operations is called a *delayed precedence constraint*. Dauzère-Pérès and Lasserre [11] were the first to incorporate delayed precedence constraints in the SB procedure. As a result, they ensure a monotonic decrease of the makespan in the bottleneck reoptimization step. They use approximation algorithms to solve the machine scheduling subproblems where the operations have release dates, run-out times, and delayed precedence constraints. Test results show that the quality of the schedules generated by this modified SB procedure is generally better than those generated by the standard SB procedure. If we want to

incorporate delayed precedence constraints in the SB procedure with extensions, then each algorithm for the machine scheduling subproblems should be changed to deal with these constraints. The computation of the delayed precedence constraints, however, takes $O(N^2)$ time, which may be too much for practical instances.

Balas et al. [2] propose an algorithm that solves the single-machine problems with delayed precedence constraints to optimality. The algorithm solves large instances that were randomly generated in a way similar to that of Carlier [7]. Balas et al. use this algorithm in the SB procedure with a modified bottleneck reoptimization step. Test results show that this variant of the SB procedure finds consistently better results than the standard SB procedure, but the computation time increases a lot. Another exact algorithm for the single-machine scheduling problem with delayed precedence constraints is proposed by Dauzère-Pérès [10].

In the standard SB procedure, the bottleneck reoptimization step consists of rescheduling the bottleneck machines one by one. Balas and Vazacopoulos [3] propose to reoptimize *partial* schedules by applying a variable-depth search algorithm. This algorithm takes about the same computation time as the algorithm of Balas et al. [2], but performs better; cf. Vaessens et al. [21].

6. Conclusions

The SB procedure has been proven to be a good method to schedule classical job shops. Those shops, however, rarely occur in practice. We showed that the SB procedure can easily be extended to deal with practical features such as transportation times, multiple resources, and down times. Experiences at a number of Dutch companies show that the SB procedure with extensions performs well. Recently, some modifications of the SB procedure have been proposed. Most modifications, however, result in large computation times and are, therefore, not usable in practical situations. The SB procedure with extensions is currently part of a commercial shop floor control system called JOBPLANNER.

Acknowledgements

The author is grateful to Steef van de Velde and Henk Zijm for helpful comments on earlier versions of this paper. Also, the author is grateful to two anonymous referees for their constructive comments.

References

[1] J. Adams, E. Balas and D. Zawack, The Shifting Bottleneck procedure for job shop scheduling, Management Science 34(1988)391–401.

[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(1995)94–109.

[3] E. Balas and A. Vazacopoulos, Guided local search with shifting bottleneck for job shop scheduling, Management Science Research Report No. MSRR-609, Carnegie-Mellon University, Graduate School of Industrial Administration, Pittsburg, PA, 1994.
[4] G.A. Belderok, Ontwerp van een produktiebesturings- en informatiesysteem voor de produktie van zware persdelen in de plaat komponenten fabriek van DAF Trucks Eindhoven (in Dutch), Master's Thesis, Faculty of Mechanical Engineering, Production and Operations Management Group, University of Twente, Enschede, The Netherlands, 1993.
[5] J.D. Blackburn, *Time-Based Competition, The Next Battle Ground in American Manufacturing*, Richard D. Irwin, Homewood, IL, 1991.
[6] P. Brucker, B. Jurisch and B. Sievers, A branch and bound algorithm for the job-shop scheduling problem, Discrete Applied Mathematics 49(1994)107–127.
[7] J. Carlier, The one-machine sequencing problem, European Journal of Operational Research 11 (1982)42–47.
[8] J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize the makespan, European Journal of Operational Research 29(1987)298–306.
[9] J. Carlier and E. Pinson, An algorithm for solving the job-shop problem, Management Science 35 (1989)164–176.
[10] S. Dauzère-Pérès, A procedure for the one-machine sequencing problem with dependent jobs, European Journal of Operational Research 81(1995)579–589.
[11] S. Dauzère-Pérès and J.-B. Lasserre, A modified shifting bottleneck procedure for job-shop scheduling, International Journal of Production Research 31(1993)923–932.
[12] H. Fisher and G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in: *Industrial Scheduling*, eds. J.F. Muth and G.L. Thompson, Prentice-Hall, Englewood Cliffs, 1963, pp. 225–241.
[13] Ph. Ivens and M. Lambrecht, Extending the shifting bottleneck procedure to real-life scheduling applications, European Journal of Operational Research 90(1996)252–268.
[14] G.J. Meester, Multi-resource shop floor scheduling, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1996.
[15] G.J. Meester and W.H.M. Zijm, Multi-resource scheduling for an FMC in discrete parts manufacturing, in: *Flexible Automation and Integrated Manufacturing*, eds. M.M. Ahmad and W.G. Sullivan, CRC Press, Atlanta, GA, 1993, pp. 360–370.
[16] I.M. Ovacik and R. Uzsoy, A Shifting Bottleneck algorithm for scheduling semiconductor testing operations, Journal of Electronic Manufacturing 2(1992)119–134.
[17] F.F.J. Reesink, Werkplaatsbesturing met behulp van de Shifting Bottleneck Methode (in Dutch), Master's Thesis, Faculty of Mechanical Engineering, Production and Operations Management Group, University of Twente, Enschede, The Netherlands, 1993.
[18] J.M.J. Schutten, Assembly shop scheduling, Technical Report LPOM-95-15, Faculty of Mechanical Engineering, Production and Operations Management Group, University of Twente, Enschede, The Netherlands, 1995.
[19] J.M.J. Schutten, Shop floor scheduling with setup times: Efficiency versus leadtime performance, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1996.
[20] 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, Management Science 42(1996)1165–1174.
[21] R.J.M. Vaessens, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by local search, to appear in Mathematical Programming.
[22] S.J. Westra, Het één-machine scheduling probleem met gefixeerde jobs (in Dutch), Master's Thesis, Faculty of Mechanical Engineering, Production and Operations Management Group, University of Twente, Enschede, The Netherlands, 1994.
[23] A.P. Woerlee, Decision support systems for production scheduling, Ph.D. Thesis, Erasmus University, Rotterdam, The Netherlands, 1991.