PRODUCTION AND OPERATIONS MANAGEMENT
Vol. 6, No. 1, Spring 1997
Printed in U.S.A.

AN EXPOSITION OF MULTIPLE CONSTRAINT SCHEDULING AS IMPLEMENTED IN THE GOAL SYSTEM (FORMERLY DISASTER™)*

JACOB V. SIMONS, JR. AND WENDELL P. SIMPSON III[†]
Department of Management, Georgia Southern University, Statesboro, Georgia 30460, *USA*
The TOC Center, Dayton, Ohio 45431, *USA*

Since Eli Goldratt first appeared on the scene in the late 1970s, his ideas concerning production management have generated a huge amount of interest, controversy, and misunderstanding. These ideas have been proliferated under several names such as optimized production technology (OPT), drum-buffer-rope (DBR), synchronized manufacturing (SM), and theory of constraints (TOC). Although there seems to be general agreement on the importance of how capacity-constrained resources are scheduled, research aimed at advancing the state of the art for the specific problem addressed by DBR continues to be limited by prior misunderstandings and the lack of a rigorous examination by the academic community. This paper seeks to advance the state of research on constraint scheduling in several ways. First, it presents a concise history of the evolution of DBR. It then explains the use of *rods* in constraint scheduling. Next, it presents in detail the solution algorithm incorporated by the Goldratt Institute in their production software and, finally, relates that algorithm to alternative methods. In the process of these activities, several lingering misconceptions are resolved.
(CONSTRAINT SCHEDULING; DRUM-BUFFER-ROPE; SEQUENCING)

1. The Evolution and State of DBR

1.1. *Past History and DBR Implementation in Software*

In 1979, Eli Goldratt, an Israeli physicist, worked with Creative Output, Inc. to produce a production scheduling software package called optimized production technology (OPT). Large corporations such as General Electric and General Motors began to report dramatic reductions in lead time and inventories attributable to OPT and its underlying philosophy (Jayson 1987). During this period, practitioners began using the term synchronized manufacturing (SM) in an attempt to show broader applicability (Umble and Srikanth 1990).

In 1984, Goldratt coauthored *The Goal* (Goldratt and Cox 1984), a landmark book that conveyed many of his fundamental ideas in the format of a novel about a harried plant manager. While *The Goal* dramatically increased public (including academic) awareness

* Received January 1996; revision received March 1996; accepted July 1996.

† Although the second author of this paper is now affiliated with the TOC Center, it should be recognized that this research was accomplished while he was a faculty member at the Air Force Institute of Technology and before his decision to change employers was ever made, discussed, or even considered. To the best of our knowledge, the research reported here was accomplished and reported objectively.

1059-1478/97/0601/003$1.25

of Goldratt's approach, the OPT software remained proprietary. Frustrated by lack of access to both his software's algorithm and his customer list (which would have facilitated case studies), many academics chose to discount Goldratt's approach and continue their focus on more traditional research problems and approaches. (Personalities also constituted a barrier to cooperative research.) A few worked with the information that was available to try to relate OPT principles to known theories (Ronen and Starr 1990).

In an attempt to shift the focus from software sales to education, Goldratt worked with Bob Fox to create the Avraham Y. Goldratt Institute (AGI) in 1987 (Fox 1994). Supported by a series of new publications (Goldratt and Fox 1986; Goldratt 1988; Goldratt 1990a, 1990b), AGI primarily sought to convey Goldratt's ideas about production management via short courses. The term drum-buffer-rope (DBR) emerged as a descriptor of the way production systems were to be visualized and managed. Five focusing steps defined how DBR was to be applied:

1. Identify the system constraint(s). A *constraint* is loosely defined as something that restricts system throughput. Its identification is usually accomplished via rough-cut capacity checking, although timing could also cause a resource to become a constraint.
2. Exploit the constraint(s). Involves always keeping the constraint producing and prioritizing *what* it produces (off-loading lower priority activities) to maximize overall system throughput. Suggests the need to create production schedules for constraint resources.
3. Subordinate all other decisions to step 2. Determines what the nonconstraint resources should be doing. In an active sense, this means doing what's necessary to support the constraint. In a passive sense, this means *not* producing more than the constraint can handle.
4. Elevate the constraint. Only after steps 1–3 does it make sense to consider increasing the capacity of the constraint.
5. Don't let inertia set in. (Iterate back to Step 1.) Actions taken in steps 3 and 4 to exploit the constraint and increase its capacity will alter the loads placed on other resources. Consequently, the system constraint(s) may have shifted.

Although fundamental DBR ideas were fairly straightforward, their implementation in full-scale production systems was facilitated by algorithms that were synthesized in a 1990 AGI-produced software package called DISASTER™. The software was given this unusual name because of Goldratt's conviction that potential users would experience disastrous consequences if they attempted to use it without understanding and applying the underlying philosophy. Unlike its predecessor OPT, the new software's logic was not considered proprietary. Goldratt himself published much of the fundamental logic (1990b) and other journal articles were published by those involved in or close to the development effort (Schragenheim and Ronen 1990; Ronen and Rozen 1992). However, probably due to the earlier unsatisfactory experiences, few academics pursued rigorous research on the topic.

Most recently, Bob Fox and others have amicably split off from AGI to form The TOC Center. The TOC Center has acquired rights to both AGI's educational products and the DISASTER™ software, which they have renamed the GOAL SYSTEM (Fox 1994) and which is maintained by the Goal Systems Group.

1.2. *Questions/Concerns/Criticisms of DBR*

For the most part, DBR remains a semi-isolated phenomenon in the operations management literature. Most introductory textbooks have now included a chapter on TOC, DBR, or OPT; however, its ideas are not well integrated with the remainder of the material presented, leading students to consider it a separate "philosophy" that can either be adopted or not, as a matter of personal preference. Even in the research literature, several lingering concerns, criticisms, and/or misconceptions remain apparent. Of particular relevance for the purposes of this paper are the notions that (1) DBR scheduling logic remains proprietary, (2) DBR constraint scheduling deals only with product mix decisions and/or requires only a simplistic timetabling of the constraint, and (3) DBR logic is capable of handling only a single constraint. There are probably two main reasons these ideas persist. For one thing, the AGI short courses and the articles and books published in the popular literature are insufficient for an in-depth understanding of the scheduling logic involved. In addition, there is a paucity of more rigorous studies by academics. This paper seeks to resolve some of these misunderstandings, thereby facilitating more productive future research and better integration of DBR ideas with other areas of production scheduling.

Specifically, the paper describes how multiple constraints can be scheduled in a DBR system and shows how such schedules are built using a commercially available software package. This may best be described as an exploratory study since it is intended to provide a foundation for further research (some of which has already been accomplished). According to Emory's (1985) classification scheme, the design is ex post facto (since no manipulation of variables was attempted) and descriptive (since it focuses on what is done and when). The study might also be characterized as a cross-sectional case study, since it focuses on a single implementation of multiple constraint scheduling logic at a particular point in time. The knowledge communicated is arguably empirical since it is based on real world observations gained through an outside observation data collection method (Flynn et al. 1990).

Fry, Cox, and Blackstone (1992) described the optimized production technology (OPT) software, a predecessor of the software explained in this research. Their study focused on the software's overall structure (modules, files, etc.) and its batch sizing calculations, which are most relevant to a make-to-stock environment. In addition to describing a more current software implementation, the present study differs by focusing on the logic of how constraint operations are sequenced to achieve order due dates and how interaction among the schedules for multiple constraints is accommodated. These issues are more relevant in a make-to-order context. In addition, we relate the subject approach to alternative methods.

In the next section, fundamentals of constraint scheduling are presented, including the DBR concept of *rods*. Subsequently, the constraint scheduling method implemented in the GOAL SYSTEM (GS) (formerly DISASTER™) is presented in detail. In the final section, the GS algorithm is characterized and related to alternative methods. In the process, it will be shown how the details of the solution method attempt to achieve the five focusing steps and the previously cited misconceptions will be resolved.

2. Constraint Scheduling and Rods

A bottleneck is defined to be any resource whose capacity is less than or equal to the demand placed on it (Umble and Srikanth 1990). Clearly, such a resource may prove to constrain the overall production system by precluding additional throughput. However, even if a resource has sufficient total capacity available to accomplish its workload during the planning horizon (if timing was ignored), the times at which jobs become available for a particular operation using that resource may preclude their completion when required by the schedule. Resources where this condition occurs have been described by some authors as "temporary" or "wandering" bottlenecks. In the TOC literature, they are termed capacity constraint resources (CCRS) (Umble and Srikanth 1990). For the remainder of this paper, the term "constraint" will be used to refer to CCRS, regardless of whether the CCR is also a bottleneck, since it is the limiting of throughput which is the primary concern.

From a drum-buffer-rope perspective, the key to the effective utilization of the overall system lies in how the constraints are managed, or *exploited*, and supported (via subordination). A schedule called a *drum* is built for each constraint to ensure the constraint will be used to its fullest effect. Since nonconstraint resources have, by definition, more than sufficient capacity to keep pace with the constraint schedule, it is considered unnecessary to generate detailed schedules for their operations. Instead, a roughly estimated amount of time, referred to as a *buffer*, is built into the schedules to allow sufficient lead time for nonconstraint operations to occur. The term "rope" refers to the lead time offset (based on the buffer size) used to generate upstream schedules. For example, a logical rope ties a constraint resource's schedule (drum) to the raw material release schedules for its gateway operations, since the release schedules will call for the release of only those materials needed to support the constraint schedule, and will offset their release times by the amount of time specified by the size of the buffer prior to the constraint.

The use of buffers in lieu of detailed schedules for nonconstraint resources is one of the distinguishing features of a DBR system. When processing times are viewed as being deterministic, specifying operation start and completion times becomes a relatively straightforward matter given the completion time of the preceding operation. Under such an assumption, it might seem reasonable to build schedules for nonconstraint operations, if for no other reason than to determine the earliest feasible start times for dependent constraint operations. However, a DBR system is based on the expectation that statistical fluctuations will be present in processing times. If that were the case, a constraint resource scheduled to begin an operation at the scheduled completion time of a preceding operation might experience idle time due to starvation. This loss of productive time on a constraint resource represents lost throughput to the overall production system. The use of buffers helps decouple operations to protect against the undesirable effects of statistical fluctuations.

The flow through a simple DBR system is relatively straightforward and has been explained elsewhere (Schragenheim and Ronen 1990; Spencer 1991). When a production system contains only a single resource that has been identified as a constraint, all timing decisions are derived from a schedule constructed specifically for that resource. Each job's release time is computed by subtracting a constant lead time from the time that job is scheduled to begin work at the constraint resource. The amount of lead time is determined by management and is designated as the size of the constraint buffer. This lead time accommodates operations required before the constraint. The job's estimated overall completion time is then computed by adding another lead time (designated as the size of the shipping buffer) to the time the job is scheduled to complete work at the constraint resource. This allows time for operations subsequent to the constraint.

Note that since the use of buffers accommodates all time required for nonconstraint operations, the constraint's schedule will compress the setup and processing operations into the smallest period of time possible. However, the problem becomes far more complex when more than one constraint operation is required. Consider initially the case where a job is processed by a constraint resource, moves on to be processed by some nonconstraints, and then returns to the original constraint resource for a different operation. The question arises as to how close in time the two constraint operations may be built into the constraint's schedule. If insufficient time is allowed for the intervening nonconstraint operations, the constraint's schedule will be disrupted due to a delay in its ability to begin the second operation.

Goldratt (1990b) introduced the notion of a *rod* in response to this problem. A rod is nothing more than a required time lag between constraint operations. A rod is built into a constraint schedule as a time spacer between constraint operations on the same job. It serves the purpose of ensuring nonconstraint operations following one constraint operation have sufficient time for completion before a subsequent constraint operation. (Although GS makes a distinction between backward rods and forward rods, the distinction is unimportant for the present discussion.) In the case of operations on the same constraint, the rod is referred to as a batch rod since it provides the required time between two process batches in the constraint's schedule. When the constraint operations involve multiple constraints, the rods are referred to as time rods since they indicate when one constraint's operation must be complete to conform to the time called for in another constraint's schedule.

The duration of a rod (i.e., the rod length) may be arbitrarily set at one-half the constraint's normal buffer size. The rod's placement in the constraint schedule is a function of two factors: transfer batch size and the relative magnitude of the per unit processing time in the two constraint operations being spaced apart. Transfer batch size refers to the number of units in a job that must be completed before they can be transferred to the next operation for processing. Ideally, transfer batches would consist of single units, since that would permit subsequent processing to begin as soon as possible. When the processing time per unit is *smallest* for the earlier constraint operation, the rod is placed between the scheduled completion of the *first* unit on the first constraint operation and the scheduled start of processing of the *first* unit on the second constraint operation. This situation is illustrated in Figure 1. Note that since the processing time per unit is smaller on the earlier constraint operation, the placement of the rod using the first unit in the batch ensures that all units will be processed at least as far ahead as the rod length before they are scheduled to be processed in the second constraint operation. By contrast, when the processing time per unit is *largest* for the earlier constraint operation, the rod is placed between the scheduled completion of the *last* unit on the first constraint operation and the scheduled start of processing of the *last* unit on the second constraint operation (see Figure 2). If the rod had been placed based on the completion of the *first* unit, less than the rod length would exist between the two constraint operations for the subsequent parts. By using the *last* unit in this case, the two processes are separated sufficiently for *all* units.

The next section explains how the GOAL SYSTEM schedules constraints using the concepts of drums, buffers, and rods.

3. The GS Algorithm Revealed

The GS method for scheduling constraints is derived directly from the first three focusing steps of identifying the constraint, exploiting the constraint, and subordinating to the constraint. Initially, the customer market is assumed to be the only constraint, i.e., no internal resource constraints are assumed. The job order schedule is then used for rough-cut capacity planning to determine the sufficiency of each resource's capacity using daily time buckets. Internal constraints are identified as resources with insufficient capacity to achieve customer due dates for the effective planning horizon. Exploitation of a constraint

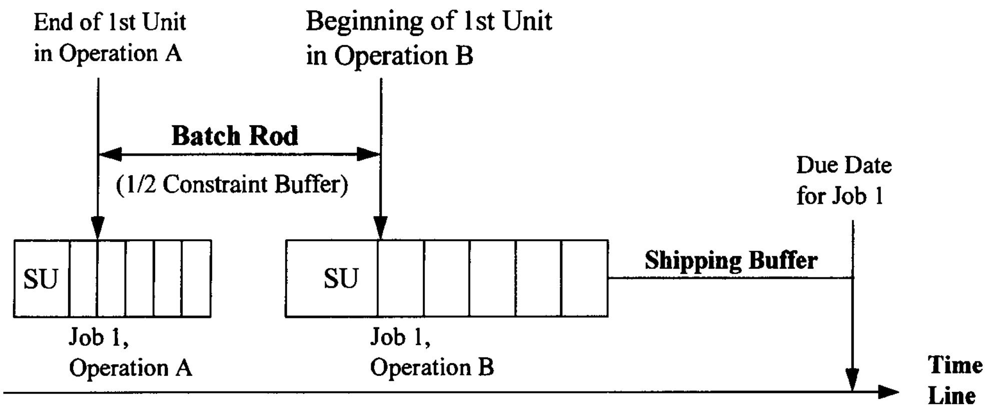


FIGURE 1. Batch Rod (Smaller Time per Part on Earlier Operation).

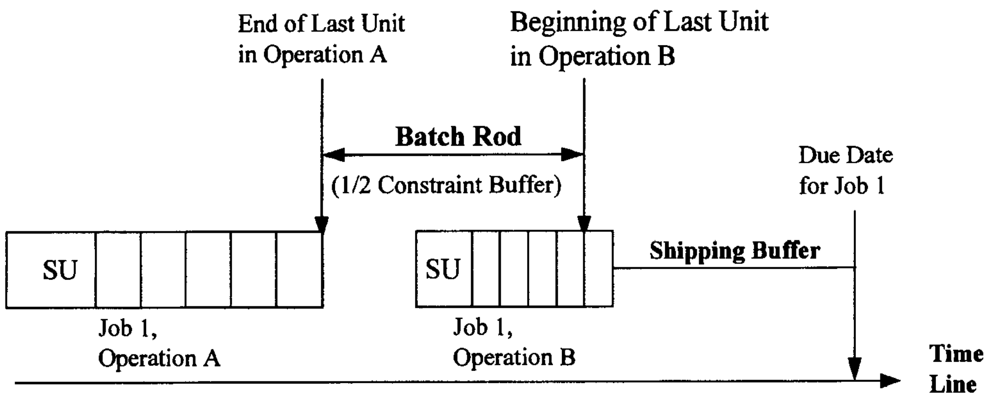


FIGURE 2. Batch Rod (Larger Time per Part on Earlier Operation).

is achieved by creating a finite schedule for the operations required of that resource. Subordination involves a return to rough-cut capacity planning for nonconstraint resources, this time in consideration of the operation due dates specified in the constraint schedule. Contrary to popular understanding (e.g., Morton and Pentico 1993), iteration among these steps is both necessary and achievable in the presence of multiple constraints. Specifically, the subordination process may reveal the existence of additional constraints, which triggers a return to the exploitation step (the scheduling of constraints).

Each of the focusing steps is accomplished via a combination of heuristics involving a series of backward and forward passes. A macro-level conceptual representation of the GS algorithm follows. (New terms are defined in the subsequent explanation and an illustrative problem is given in the Appendix.)

Step 1: Compute effective horizon = planning horizon + shipping buffer

Step 2: Subordinate all resources to the market

Step 2a: Working backward in time from the end of the effective horizon, calculate daily loads for each resource to achieve all job due dates, assuming jobs are backward scheduled

Step 2b: Identify resource constraints via first day load (FDL) peaks

Step 2c: If no resource constraints are identified, go to step 7

Step 3: Build drum schedule for primary constraint resource

Step 3a: Build *ruins* (a Gantt chart that ignores capacity) using operation finish time = job due date − shipping buffer and including any batch rods required due to multiple constraint operations for the same job

Step 3b: Accomplish backward pass to level the ruins consistent with available capacity of constraint resource

Step 3c: If batches are scheduled in the past, accomplish forward pass to achieve feasibility (Note: At this stage, the user is given the opportunity to specify actions that would further exploit the constraint, e.g., combining batches to avoid setups, offloading operations to other resources or to subcontractors, etc.)

Step 3d: Fix drum schedule in time and reconcile constraint batch times with order due dates

Step 4: Subordinate nonconstraint resources to the market and the drum schedule(s)

Step 4a: Reaccomplish backwards rough-cut capacity check for nonconstraint resources (as in step 2a) to satisfy both job due dates and constraint operation due dates

Step 4b: Identify additional constraints via the presence of either FDL peaks or red lane (RL) peaks

Step 4c: If no additional constraints are identified, go to step 7

Step 5: Build schedule for additional drum

Step 5a: Build drum schedule as in steps 3a–3d, but consider time rods from any fixed constraint schedules, as well as batch rods for the additional constraint
Step 5b: Identify drum violations (infeasibility due to overlaps of the additional drum's schedule with time rods for other existing constraint schedules) (Note: At this point, the user again has the opportunity to manipulate the schedule, this time to resolve drum violations.)
Step 5c: If no drum violations exist, go to step 4
Step 6: Drum loop
Step 6a: Rebuild the first fixed constraint schedule, shifting batches later by the amount of the drum violation
Step 6b: Unfix (eliminate) all additional constraint schedules
Step 6c: Go to step 4
Step 7: Stop. Implement drum schedules.

Step 1 expands the planning horizon to incorporate a shipping buffer, which is an additional period of time allocated for performing any operations that must be accomplished following constraint processes but before the job's due date. The shipping buffer decouples constraint operations from order receipt by the customer, thereby protecting our ability to meet the due date. By adding the shipping buffer to the horizon, we ensure planning is accomplished for jobs whose constraint processing must be completed during our intended scheduling horizon so that the shipping buffer before their order due dates is not "eaten into." Step 2 is an attempt to determine whether the customer orders can be met without treating any resources as internal constraints. At this stage, only the customer due dates themselves are being treated as constraints. If all resources have sufficient capacity to support the due dates, then no constraint schedules will be necessary (other than the shipping schedule itself).

Step 2a works as follows. Beginning with the latest due date, each job's operational requirements are subtracted from the remaining capacity of each affected resource. If a resource's capacity for that day is exceeded, the required load is moved back a day and all preceding operations will be checked using this earlier date. Although step 2a employs only rough-cut capacity checking, GS allows users to set effective capacity levels at some level lower than the apparent capacity to accommodate timing complications within the work day. When resources have excessive loads, the net effect will be an accumulation of workload flowing backward (earlier and earlier). Ultimately, the process will show that such resources may have loads on the first day greater than their available capacity. Such a condition is referred to as a FDL peak and reveals the presence of a constraint resource.

If more than one resource has a FDL peak, GS recommends the one with the largest peak for step 3. The term "ruins" in step 3a can best be visualized in Gantt chart form. Figure 3 shows a simple illustration of the ruins for a single constraint resource with two jobs to process. Each job requires two operations on the constraint resource with the need to visit other (nonconstraint) resources in between the two constraint operations. Each job is situated on the timeline such that the processing of its final operation would be completed exactly one shipping buffer's length of time before the job due date. Since each job requires multiple operations on the same resource, batch rods are used in lieu of the due date and shipping buffer to determine placement of the earlier operations on the timeline. The batch rods permit the time necessary for nonconstraint operations that must occur between the two constraint operations. Note that the setup and processing time for the jobs has been laid on the timeline without regard to available capacity. If the number of machines available is less than the number of layers needed to represent the ruins, the schedule is clearly infeasible.

In step 3b, the constraint schedule is made capacity-feasible by working backward, left-shifting operations as required to permit operations to "drop down" on the Gantt chart

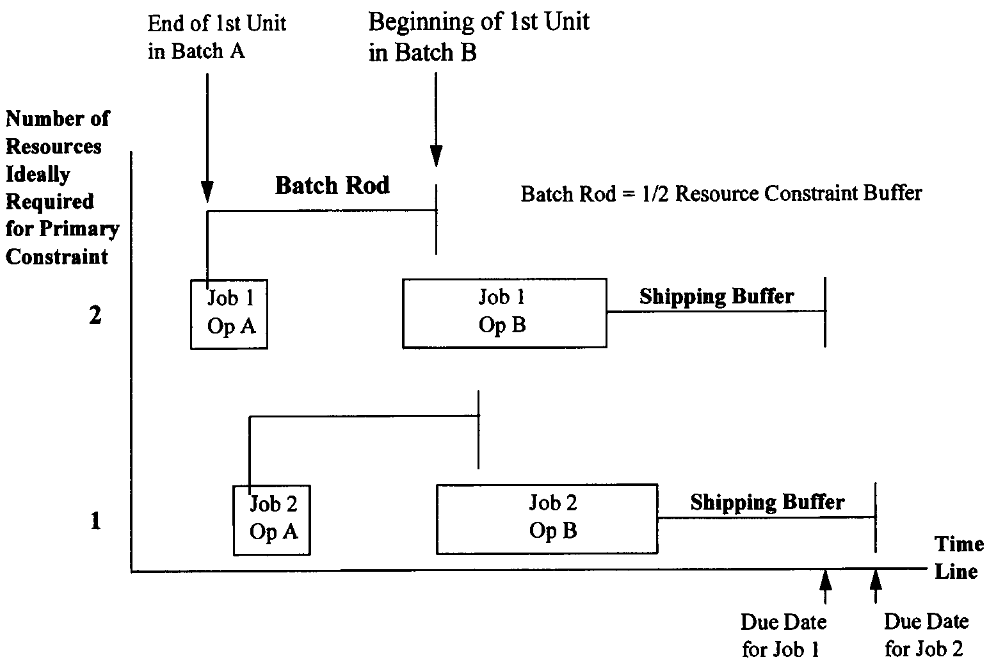

End of 1st Unit in Batch A
Beginning of 1st Unit in Batch B
Number of Resources Ideally Required for Primary Constraint
Batch Rod
Batch Rod = 1/2 Resource Constraint Buffer
2
Job 1 Op A
Job 1 Op B
Shipping Buffer
1
Job 2 Op A
Job 2 Op B
Shipping Buffer
Time Line
Due Date for Job 1
Due Date for Job 2

FIGURE 3. Ruins for a Single Constraint with Two Jobs (James and Mediate 1993).

consistent with the number of machines available (see Figure 4). The net result will be that the operations will remain sequenced in order of the ideal completion times first shown in the ruins, but will be spread out over a longer period of time. Since the left-shifting operation has moved operations earlier in time, it is likely that this backward pass will result in some operations being scheduled to occur in the past. This infeasibility is resolved by the forward pass of step 3c, which shifts the entire schedule as far right as necessary to avoid work being scheduled in the past (see Figure 5). Since this will likely cause some jobs to have less than the desired shipping buffer or to exceed their customer due dates outright, step 3d includes an opportunity to revise promised order dates.

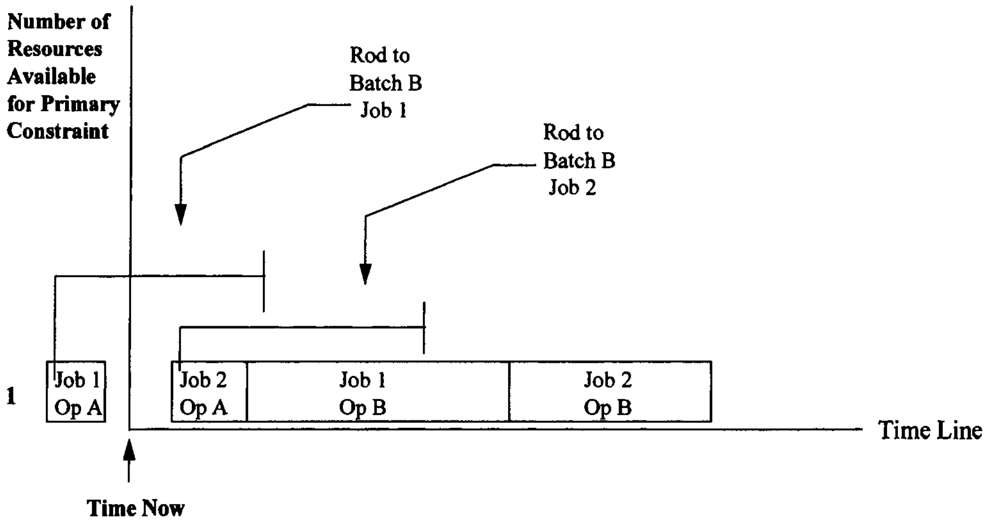


FIGURE 4. Schedule for Batches after Backward Pass (James and Mediate 1993).

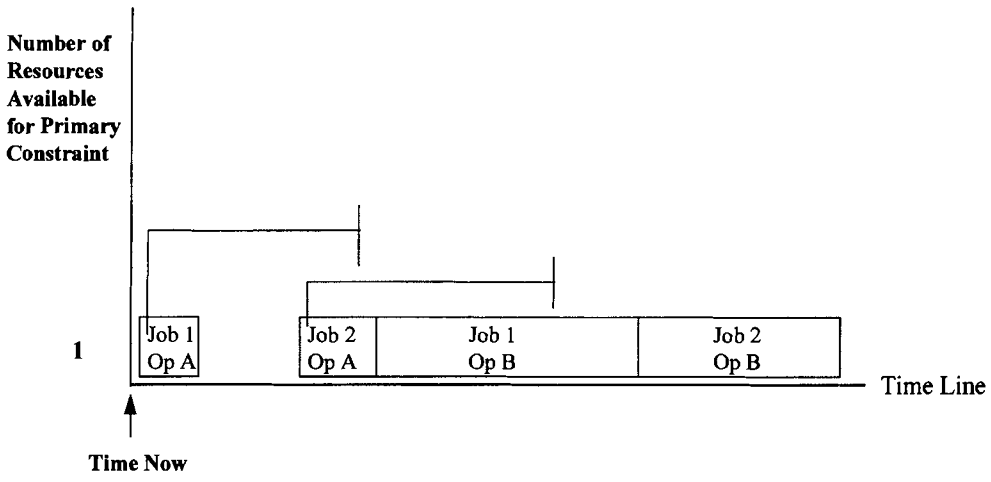


FIGURE 5. Drum Schedule after Forward Pass (James and Mediate 1993).

Having created a schedule for the constraint resource (exploiting the constraint), the GS algorithm revisits the notion of subordination in step 4. Specifically, rough-cut capacity checking will be reaccomplished for the nonconstraint resources. Step 4a differs from step 2a in that the dates on which workload is allocated to the nonconstraints are now determined by both the original job due dates *and* the operation *due dates* required to support the constraint schedule. Therefore, nonconstraint operations that occur *subsequent* to the constraint operations are evaluated for capacity in time to achieve the overall job due date, while those that must occur *before* a constraint operation are evaluated for capacity in time to meet the schedule of the constraint. Infeasibility in the former case is referred to as a RL peak, while infeasibility in the latter case retains the FDL peak designation used in step 2.

As was the case in step 2, it is possible that additional resources will not have sufficient capacity when checked in step 4. However, it is not necessarily the case that the same resources will appear to be overloaded in step 4. If, for example, the CCR schedule built in step 3 has the net effect of pushing overall job completion dates farther into the future, resources that were overloaded in step 2 may now have the additional time necessary to achieve the revised due dates. Conversely, step 3 has generated additional timing requirements to support the drum schedule. These may not be supportable by resources which previously appeared (in step 2) to be nonconstraints. It is the combination of these two possibilities which necessitates the resubordination of step 4.

If step 4b identifies any additional constraints, the most heavily loaded one is now scheduled in step 5. The logic of step 5 is similar to that of step 3. However, the schedule to be built for the additional constraint (step 5a) must also be supportive of the schedule built in step 3 for the primary constraint resource. This is accomplished by the use of time rods, which are placed on the timeline where necessary to ensure that sufficient buffer time is provided between the two constraint schedules. As before, batch rods must also be inserted to facilitate any nonconstraint processing that must occur between multiple operations for the same job on the additional constraint. A drum violation (step 5b) occurs whenever the time rods generated by the primary drum schedule are too restrictive to permit the generation of a feasible schedule for the additional constraint.

If no drum violations are encountered, the GS algorithm returns (step 5c) to step 4 and rechecks the nonconstraint capacities. However, a drum violation (step 5b) necessitates the drum loop in step 6. A drum loop is no more than the rebuilding of the schedule for the primary constraint during which constraint operations are shifted later in the schedule by the amount of the drum violation to avoid the detected infeasibility. (The constraint operations are shifted only later to reduce the likelihood of FDL peaks during iteration and to provide more room for placement of batches on later drums.) Since the primary constraint is being rescheduled with more time allowed to do the work, it is now possible that previously identified additional constraints may no longer have excessive workloads. To check this possibility, all additional constraint schedules are discarded (step 6b) and subordination is reaccomplished (step 6c).

Note that iteration among steps 4–6 could occur multiple times, depending on the impact of drum violations on the ability of other resources to support any drum schedules that have been built and retained. As this occurs, different resources may alternately gain and lose designation as constraints. The end result is that CCR schedules will be produced for one or more resources and all remaining resources will have been determined to have sufficient capacity during the scheduling horizon to support the CCR schedules. This invalidates one of the common misconceptions concerning the TOC/GS solution approach. The examples used in introductory courses and articles typically illustrate cases in which only a single constraint exists. Consequently, it appears that OPT-like scheduling methods apply only to single constraint cases (Morton and Pentico 1993, p. 28). This is clearly not true of the GS solution algorithm.

Since the GS algorithm involves the insertion of rods and the use of forward passes (right-shifting) to avoid work in the past, it is likely that some job due dates will not be achieved by the final constraint schedule(s). Following step 7, GS displays the revised projected due dates of jobs that would result from implementation of the constraint schedules. The planner is given the opportunity to combine additional orders of like jobs to achieve further avoidance of setups or to manually revise the due dates for certain jobs and regenerate the schedule. (The GS software contains additional schedule-enhancing heuristics that are beyond the scope of this paper.) Each step of this algorithm is illustrated in the Appendix.

4. Characterization of the GS Scheduling Method and Its Relation to Alternative Methods

The GS algorithm may be considered a "loose" approach to production scheduling since it builds timetables only for resources designated as constraints. A raw material release schedule is derived from the constraint schedules (for jobs processed by a constraint) and the shipping schedule (for those jobs not processed by a constraint). In general, other resources are simply told to work on jobs that come to them as they arrive. (The exception is that schedules may also be built to resolve conflicts among different resources that use a common output from an upstream operation.)

The description in the previous section shows that the GS algorithm involves a fairly sophisticated amalgamation of heuristics and iteration, rather than being the single-pass, simplistic heuristic typically represented in the literature. It is, nonetheless, heuristic. The uses of rods, shipping buffers, and rough-cut capacity checking for nonconstraint resources, for examples, are hedges against infeasibility rather than guarantees.

The presence of multiple constraints leads to the use of additional heuristic logic, particularly when the constraints are interactive (i.e., some jobs require processing on more than one constraint resource type). GS uses a sequential approach to build constraint schedules. As long as a secondary constraint's schedule is feasible with respect to a primary constraint's schedule, the solution is accepted. However, it is clearly possible that an improved result might be possible if the two constraint schedules were developed simultaneously. Even the order in which multiple constraints are scheduled could affect the quality of the solution. This recognition is particularly interesting in light of the fact that GS gives the scheduler the option of picking which resource to schedule next when more than one constraint is identified during a subordination phase (step 2 or 4). According to Goal System Group personnel, companies using GS typically schedule constraints in the same order every time they produce schedules (Rose 1993). Their choice may well affect the quality of the resulting schedule, a possibility we have explored and reported separately (Simons et al. 1996).

4.1. *CONWIP*

CONWIP (Spearman, Woodruff, and Hopp 1990) is an alternative to the DBR approach that might be considered even "looser" than the GS method. In a CONWIP system, schedules are not even built for constraint resources. Instead, raw material release is controlled by the use of a limited number of work authorization cards that become available for reuse only as jobs are completed. In this manner, the total amount of work released to the floor is controlled at a rate determined by the pace of the overall system.

While kanban cards in a JIT system flow only between sequential work processes, CONWIP authorizations travel with a job the entire length of the production flow. Consequently, while the total released work is controlled, its location within the system is not. The advantage over a JIT system is that by not controlling the location of authorizations, inventory is allowed to accumulate in front of constraints (which permits them to remain busy), but is not necessarily accumulated between nonconstraint operations. The net result may be reduced overall work-in-process (WIP).

CONWIP's advantage over the GS approach is that it does not require the *a priori* identification of constraint resources and therefore avoids the need to solve the constraint scheduling problem described in this paper. Work will naturally buffer constraints that exist in the system and the constraints are free to shift dynamically. However, CONWIP implementation has been generalized only to flowshops. Extension to more complex systems is in progress, but potentially problematic (Spearman et al. 1990). In addition, it may be more difficult to recognize unplanned delays in a CONWIP system since there are no detailed resource schedules and constraint buffer management (Schragenheim and Ronen 1990) is not supported.

4.2. *Linear Programming*

Widely used for a variety of applications in the field of scheduling, linear programming (LP) has been suggested as being useful for exploiting constraints (Luebbe and Finch 1992). Its applicability is most apparent in the context of determining product mixes, rather than timetabling. This is probably the case for several reasons. First, product mix problems are traditionally recognized as being amenable to solution via LP. Second, LP provides optimal solutions while the five TOC focusing steps rely on the use of heuristics. Third, a perception exists that TOC cannot recognize or accommodate multiple constraints (Morton and Pentico 1993). If this were true, the generation of product mixes in consideration of only a single constraint might produce schedules that are infeasible due to the presence of additional constraints (Plenert 1993).

The perception that the focusing steps reveal only a single constraint is probably a result of the fact that introductions to the steps almost invariably demonstrate cases with only one constraint. In such cases, the first two of the five focusing steps appear to be sufficient to schedule the constraint. Consequently, most of those introduced to the theory conclude that these are the only two steps needed. However, as the preceding section of this paper has shown, the GS solution method employs the first *three* focusing steps (Goldratt 1990b, p. 187). It is the third step of subordination that reveals the presence of additional constraints and leads to the iterative process of scheduling multiple constraints.

Plenert (1993, p. 132) acknowledges that multiple constraints can exist, but observes LP will produce a direct, feasible solution without the need for the iteration employed by GS (nee DISASTER™). However, it must be recognized that when it is used to solve only the product mix problem, LP ignores timing, which GS does not. Consequently, while the GS solution will be feasible (to the extent that the buffers have been sized sufficiently), the LP solution may well prove infeasible when timing is considered. This is not an unlikely possibility, since LP will seek to maximize the use of the constraining resources, leaving no room for any time lags arising from precedence relationships.

4.3. *The Adams, Balas, and Zawack Shifting Bottleneck Procedure*

The GS scheduling algorithm is most closely paralleled by the shifting bottleneck procedure, which was devised by Adams, Balas, and Zawack (1988) to minimize makespan in a job shop problem. The shifting bottleneck method sequences machines one at a time. At each step, the machine selected next for sequencing is the one having the largest makespan on a relaxed, single-machine problem. (This value serves as an indicator of the extent to which any machine represents a potential bottleneck.) Each time another machine is sequenced, those previously sequenced are locally reoptimized.

Both the shifting bottleneck and GS solution methods schedule potential constraints, consider the impact on other resources, and iteratively reschedule as other potential constraints are identified. However, the shifting bottleneck procedure seeks to minimize makespan while GS focuses on due dates. (It is interesting to note at this point that GS does not formally identify a specific objective function. However, it is clear that the algorithm seeks good performance with respect to due dates.)

The shifting bottleneck method goes farther than GS by scheduling *all* resources, including those unlikely to be constraints. While this increases the comprehensiveness of the shifting bottleneck method, TOC advocates might consider it unnecessary and requiring additional computational expense, which might be substantial for large production systems. The authors' description makes it apparent that a truncated version of the method might be devised using a threshold to limit the determination of whether a resource is a sufficiently likely constraint to merit its being scheduled.

4.4. *Demeulemeester and Herroelen's GRCPSP Solution Procedure*

The constraint scheduling problem discussed here is nearly subsumed by Demeulemeester and Herroelen's general production scheduling problem (GPSP) (Demeulemeester and Herroelen 1996). Since they formulated the GPSP problem as a translation of the general resource constrained project scheduling problem (GRCPSP), they were able to apply their previously devised branch and bound solution algorithm (Demeulemeester and Herroelen 1992) to GPSP examples where the objective was to minimize makespan.

Demeulemeester and Herroelen's computation times are extremely fast. Furthermore, their branch and bound method produces optimal results. This would seem to suggest great potential for improved solutions to large sized constraint scheduling problems. However, since DBR constraint scheduling problems will typically involve a due date-related objective function, new bounds and branching rules would presumably be required. In addition, their solution method relies on an assumption of project scheduling (no successor activity can start before any of its predecessors) that is not strictly applicable to production scheduling scenarios. (It is possible that small transfer batches could permit consecutive operations to be scheduled so close together in time that the long setup of a successor operation might be scheduled to begin before the shorter setup of the predecessor operation.)

4.5. *Other Methods*

Clearly, any of a variety of other solution methods might be applicable to the constraint scheduling problem. One of our purposes in presenting this paper is precisely to facilitate and encourage such efforts. In research reported elsewhere (Simons et al. 1996), we show results that we obtained using a variation of Florian, Trepant, and McMahon's (1971) branch and bound method that, like the shifting bottleneck procedure (Adams, Balas, and Zawack 1988), is based on a graph-theoretic representation of the problem.

Since this research's initiation, development work has continued on the GS software. Although we cannot give a comprehensive list of those changes, we are at least aware that steps have been taken toward inclusion of automatic offloading to alternate resources and the need for multiple resources per operation. These revisions would need to be incorporated as other solution methods are considered.

5. Conclusions

We have presented an overview of the evolution of DBR and have highlighted some lingering misconceptions. We have explained the notion of a rod and its relation to constraint scheduling. We have also presented the GS algorithm and described its relationships with alternative scheduling methods. In the process of these discussions, we have attempted to resolve some of the concerns about the constraint scheduling problem and the GS solution method which are not valid. The misconceptions and our responses may be summarized as follows:

Misconception 1: DBR scheduling logic remains proprietary.

Response: This was true of OPT, but is not true of GS, which is now the state-of-the-art software implementation of DBR scheduling.

Misconception 2: DBR constraint scheduling deals only with product mix decisions and/or requires only a simplistic timetabling of the constraint.

Response: Introductory courses and articles frequently present only product mix decisions or cursory scheduling examples due to time and space constraints. Timetabling is also required, is potentially complex, and may involve iterative application of the first three focusing steps (identification, exploitation, and subordination).

Misconception 3: DBR logic is capable of handling only a single constraint.

Response: The GS implementation of DBR logic uses the subordination step to reveal the presence of additional constraints, which are then scheduled. Furthermore, the iterative nature of the process may cause initial constraint schedules to be revised to accommodate interaction with other constraints.

At the outset, we expressed the hope that this research would facilitate more productive future research. Specifically what research do we see the need for?

1. Evaluation of the quality of schedules produced by GS. How good are GS schedules with respect to relevant management criteria? (e.g., Simons et al. 1996) Do other commercially available packages produce schedules that facilitate better due date performance? Also, since GS relies on buffers as a means of ensuring the achievability of its schedules, are GS-produced schedules more robust to disruptions than those of its competitors? What are the consequences and/or benefits of not scheduling nonconstraints? How sensitive are GS schedules to the choice of buffer sizes?

2. Implementation considerations. How easy is GS to implement, compared to its competitors? How well does it satisfy user needs and desires?

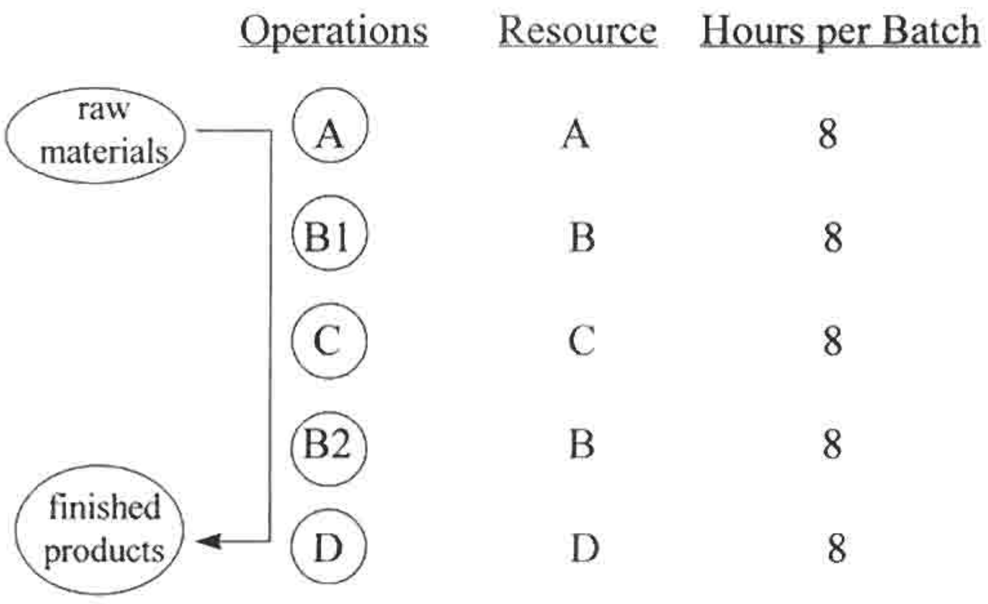
3. Algorithmic opportunities. We have pointed out some general relationships between GS and other methods, but more specifically, what logic can be adapted from other algorithms to improve GS? Conversely, what GS logic can be used to improve other algorithms?[1]

[1] The authors thank Dick Moore, Rob Newbold, Coral Rose, and the TOC Center's Goal Systems Group for their assistance in various stages of this research.

Appendix: Illustrative Problem

Most shops have greater complexity and more jobs to be scheduled than are depicted in this example. However, we beg the readers' indulgence in accepting this relatively unrealistic scenario for the greater good of its ability to facilitate understanding of the logic presented in the body of the paper. Although simplistic, the problem invokes each step of algorithm defined earlier.

The example shop receives orders for a single product type with proposed due dates. Each order is for a batch of 100 units. Each batch is processed using the sequence of operations shown:



Operations	Resource	Hours per Batch
A	A	8
B1	B	8
C	C	8
B2	B	8
D	D	8

We will use the notation J1(A) to designate the operation performed by resource A on Job J1. Since resource B performs two operations on each job, they will be differentiated as J1(B1) and J1(B2).

There is one unit of each resource type and production occurs 8 h each day, Monday through Friday. The shipping and constraint buffer sizes have been established as 8 and 16 h, respectively. The shop uses a transfer batch size of one unit (i.e., each unit can be moved to the next operation without waiting for completion of the entire batch). No setup time is required for any of the operations and raw materials are always available for each job. There is no work-in-process currently in the shop, so any jobs scheduled will have to be initiated "from scratch." Finally, we assume that the scheduler running the OS software makes no manual interventions, so that the schedule(s) produced will be those created solely on the basis of the OS algorithm.

Step 1: Compute effective horizon = planning horizon + shipping buffer.

The scheduler desires to produce a schedule for the coming week, beginning on Monday morning. Only two jobs call for completion during the planning horizon plus eight hours (one shipping buffer). Job J1 is due at hour 28 (the middle of the day on Thursday) and job J2 is due at hour 32 (the end of the day on Thursday).

Step 2a: Working backward in time from the end of the effective horizon, calculate daily loads for each resource to achieve all due dates, assuming jobs are backwards scheduled.

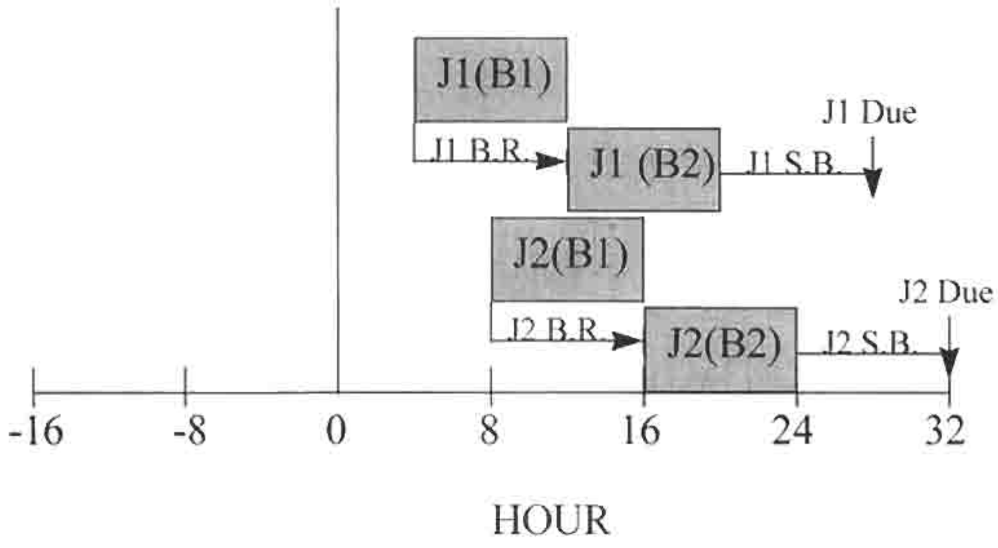
Resource		Mon	Tue	Wed	Thu	Fri
A	J1(A)		J2(A)			
B	J1(B1)	J1(B2)	J2(B1)	J2(B2)		
C		J1(C)		J2(C)		
D			J1(D)	J2(D)		

Step 2b: Identify resource constraints via FDL peaks.

To meet the proposed due dates, both resources A and B would have to have completed work in the past, which they have not done. Consequently, they would be unable to support the current plan and are identified as potential constraints. Since B has the heavier total load, we choose to select it as our primary constraint.

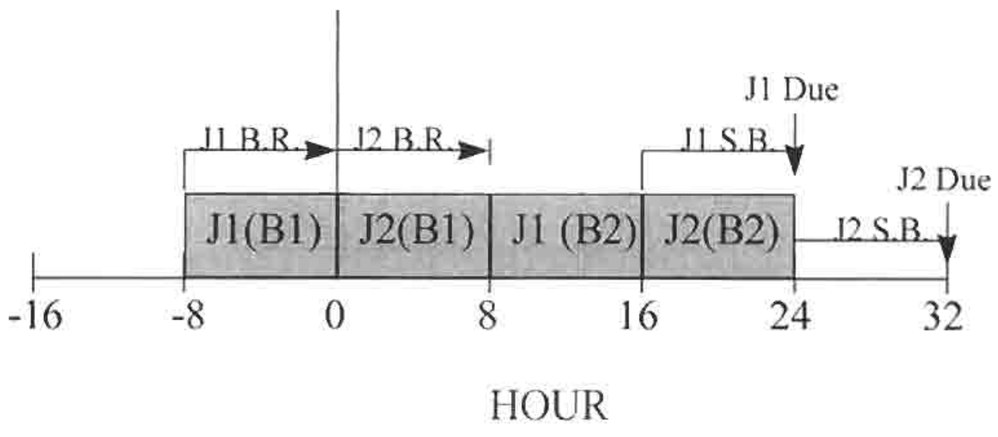
Step 3a: Build ruins for the primary constraint.

Since resource B has two operations to perform on each job and since we have specified the constraint buffer size as 16 h, we place an 8-h batch rod (BR) between the two B-resource operations on each job. The absence of setup times and the equal processing time (8 h) for each of the two B-resource operations means that we can view these rods as a required minimum 8-h time lag between the start of the first operation, e.g., J1(B1), and the start of the second operation, e.g., J1(B2), on each job. Accordingly, in consideration of both the shipping buffer and the batch rods, the ruins for B appear as follows:



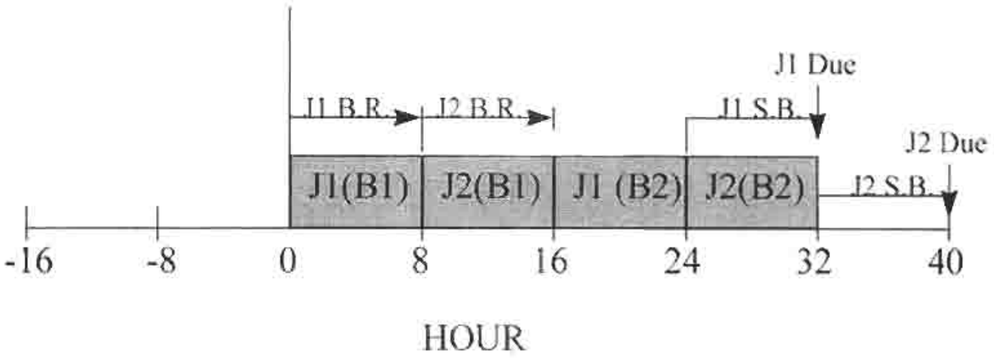
Step 3b: Accomplish backward pass to level the ruins consistent with available capacity.

Based on the proposed job due dates and the batch rods, the operations in this case would be sequenced in a manner consistent with their operation due dates.



Step 3c: If batches are scheduled in the past, accomplish forward pass to achieve feasibility.

Since operation J1(B1) is scheduled in the past, the entire sequence is right-shifted.

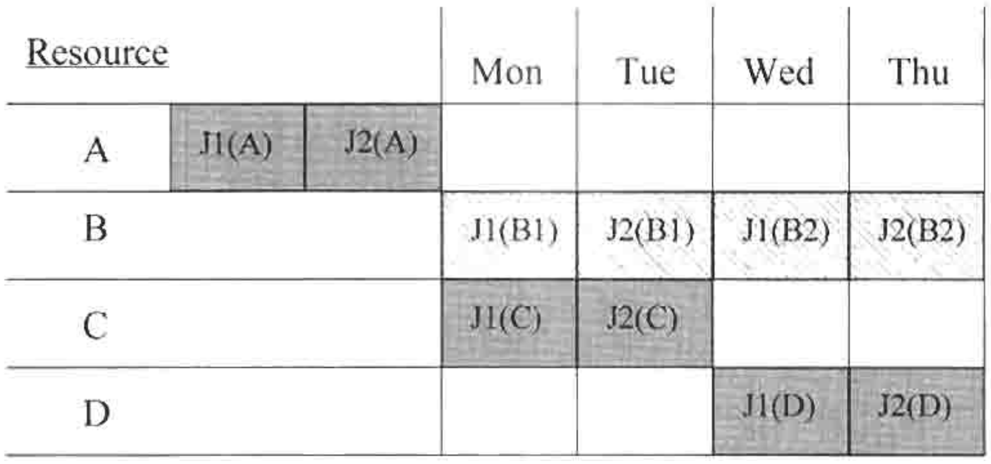


Step 3d: Fix drum schedule in time and reconcile constraint batch times with order due dates.

Based on the drum we have just created, job J1 would be completed by the constraint at hour 24. Adding the 8-h shipping buffer, the job should not be promised to the customer until hour 32. Similarly, job J2's due date would need to be changed to hour 40.

Step 4a: Reaccomplish capacity check for nonconstraint resources considering both job due dates and constraint operation due dates.

Resource D's operations are checked for capacity based on the revised job due dates. However, the operations for resources A and C are checked for capacity one constraint buffer (16 h) before when the jobs must be processed by resource B according to its schedule (although the operations on resource C will not be scheduled any earlier than the preceding B operation allows).



Step 4b: Identify additional constraints.

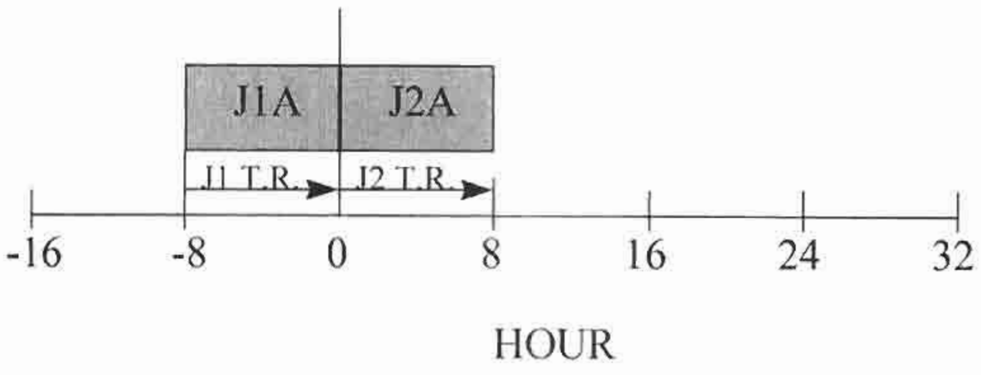
Resource A would not be able to support B's drum (schedule), since it would have to accomplish processing of job J1 before time zero. Therefore, A will also need to be treated as a constraint.

Step 4c: If no additional constraints are identified, go to step 7.

Since A has been identified as a secondary constraint, we proceed to step 5.

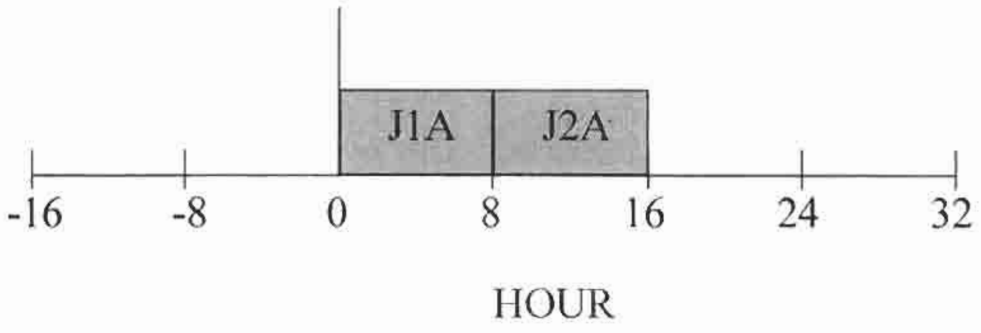
Step 5a: Build schedule for additional drum, considering time rods from previous drums.

Two time rods will need to be considered in building a drum for resource A. Operation J1(A) should not be scheduled to start less than 8 h before the start of J1(B1) (currently scheduled for hour 0) and J2(A) should not start less than 8 h before J2(B1) (currently scheduled for hour 8). Based on the scheduled start time of these operations in the current drum for B, the following schedule shows the latest the A operations can start without violating the time rods.



Step 5b: Identify drum violations.

Since we cannot schedule work in the past, we would need to push operation J1(A) forward by 8 h, which pushes J2(A) forward by the same amount.



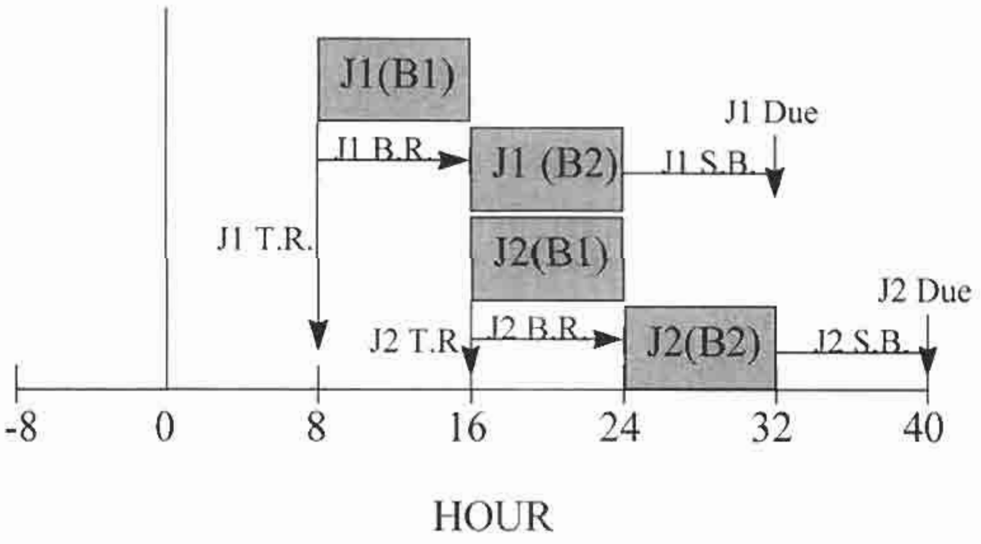
However, this violates both of the time rods between the operations on resource A and those on operation B. Therefore, we have a drum violation by an amount of 8 h.

Step 5c: If no drum violations exist, go to step 4.

Since a violation does exist, we proceed to step 6.

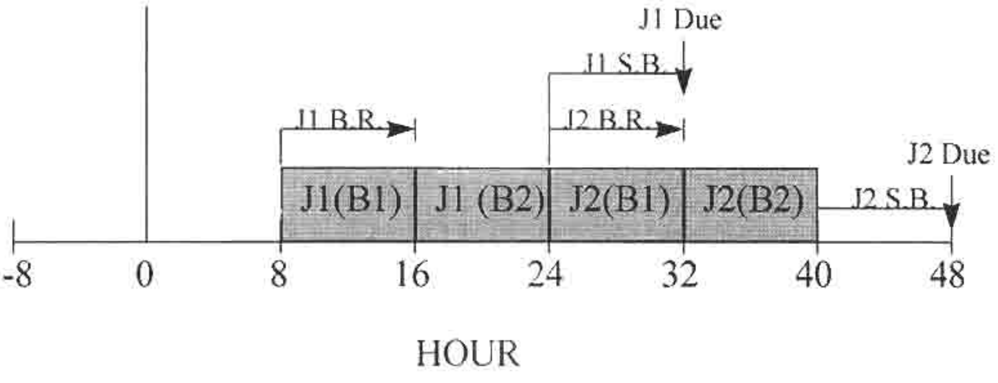
Step 6a: Rebuild the earlier constraint schedule, shifting batches by the amount of the drum violation.

Since our violation was for 8 h and affected both operations J1(B1) and J2(B1), we rebuild resource B's drum with the requirement (via time rods) that J1(B1) not start before hour 8 and that J2(B1) not start before hour 16. As before, we respect batch rods (8 h) between the first and second B-resource operation on each job.



Note that in leveling these ruins, we have to decide whether J1(B2) should be sequenced before or after J2(B1).

In this case, the choice will be made to put J1(B2) first. The reason is that, by so doing, job J1 will be completed earlier without affecting the completion time of job J2. After leveling the ruins and right-shifting to honor the time rods at hours 8 and 16, we have the following:



Based on the revised B drum, J1's due date could remain unchanged at hour 32, but J2's due date would slip from 40 to 48.

Step 6b: Unfix (eliminate) all additional constraint schedules.

Since we are unaware how the revision of B's drum would affect other constraint schedules, we return resource A to an undetermined status and delete its drum.

Step 6c: Go to step 4.

Step 4a: Reaccomplish capacity check for nonconstraint resources considering both job due dates and constraint operation due dates.

Based on the revised drum schedule and, as before, taking into account the requirement for constraint buffers before operations scheduled for resource B, we find these daily capacity loads.

Resource		Mon	Tue	Wed	Thu	Fri
A	J1(A)		J2(A)			
B			J1(B1)	J1(B2)	J2(B1)	J2(B2)
C			J1(C)		J2(C)	
D				J1(D)		J2(D)

Step 4b: Identify additional constraints.

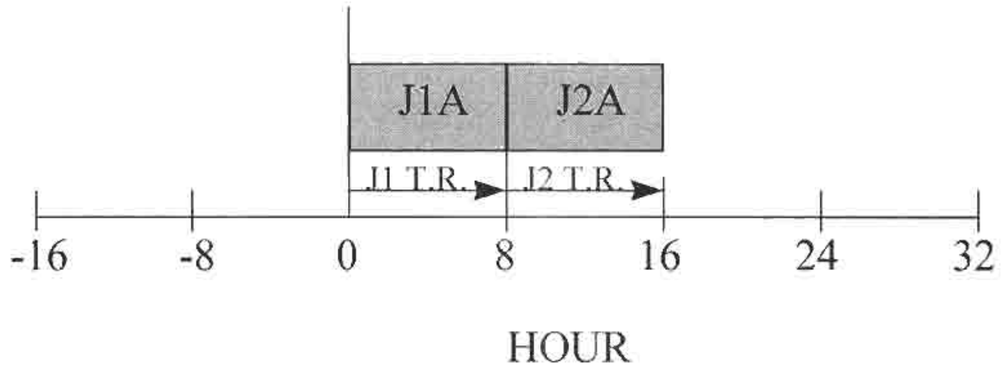
If treated as a nonconstraint resource, A would still not be able to support B's drum, since it would have to accomplish processing of job J1 before time zero. Therefore, A will again need to be treated as a constraint.

Step 4c: If no additional constraints are identified, go to step 7.

Since A has been identified as a secondary constraint, we proceed to step 5.

Step 5a: Build schedule for additional drum, considering time rods from previous drums.

As before, we build a drum for A in consideration of 8-h time rods between its operations and the first operation for each job on resource B. Note that the placement of A operations is different this time because B's operations have been slipped.



Step 5b: Identify drum violations.

This time, we have no violations. Although it was necessary to reidentify A as a secondary constraint, the lesser requirement of the time rod (8 h) vs. a full constraint buffer (16 h) makes A's constraint schedule feasible.

Step 5c: If no drum violations exist, go to step 4.

Step 4a: Reaccomplish capacity check for nonconstraint resources considering both job due dates and constraint operation due dates.

Based on the drums for both A and B and, as before, taking into account the requirement for constraint buffers before operations scheduled for resource B, we find these capacity loads.

Resource	Mon	Tue	Wed	Thu	Fri
A	J1(A)	J2(A)			
B		J1(B1)	J1(B2)	J2(B1)	J2(B2)
C		J1(C)		J2(C)	
D			J1(D)		J2(D)

Step 4b: Identify additional constraints.

This time, no additional constraints are identified.

Step 4c: If no additional constraints are identified, go to step 7.

Step 7: Stop. Implement drum schedules.

In this case, we will have two such schedules: one for A and one for B. In this simple example, the schedule for A may seem unnecessary since it has such a light load. However, it must be remembered that nonconstraint resources in a DBR system are only loosely controlled. The constraint schedule for A helps ensure that work on J1(A) will begin promptly at time zero, since there is only an 8-h buffer decoupling it from J1(B1).

References

ADAMS, J., E. BALAS, AND D. ZAWACK (1988), "The Shifting Bottleneck Procedure For Job Shop Scheduling," *Management Science*, 34, 3, 391–401.

DEMEULEMEESTER, E. L. AND W. S. HERROELEN (1992), *A Branch-And-Bound Procedure For The Generalized Resource-Constrained Project Scheduling Problem*, Research Report N 9206, Department of Applied Economic Sciences, Katholieke Universiteit Leuven, Belgium.

——— (1996), "Modelling Setup Times, Process Batches And Transfer Batches Using Activity Network Logic," *European Journal of Operational Research*, 89, 2, 355–365.

EMORY, C. W. (1985), *Business Research Methods*, 3rd ed., Irwin, Homewood, IL.

FLORIAN, M., P. TREPANT, AND G. MCMAHON (1971), "An Implicit Enumeration Algorithm For The Machine Sequencing Problem," *Management Science*, 17, B782–B792.

FLYNN, B. B., S. SAKAKIBARA, R. G. SCHROEDER, K. A. BATES, AND E. J. FLYNN (1990), "Empirical Research Methods In Operations Management," *Journal of Operations Management*, 9, 2, 250–284.

FOX, R. E. (1994), Personal correspondence.

FRY, T. D., J. F. COX, AND J. H. BLACKSTONE, JR. (1992), "An Analysis And Discussion Of The Optimized Production Technology Software And Its Use," *Production and Operations Management*, 1, 2, 229–242.

GOLDRATT, E. M. (1988), "Computerized Shop Floor Scheduling," *International Journal of Production Research*, 26, 3, 443–445.

——— (1990a), *What Is This Thing Called Theory Of Constraints And How Should It Be Implemented?* North River Press, Croton-on-Hudson, NY.

——— (1990b), *Sifting Information Out Of The Data Ocean: The Haystack Syndrome*, North River Press, Croton-on-Hudson, NY.

——— AND J. COX (1984), *The Goal: A Process Of Ongoing Improvement*, North River Press, Croton-on-Hudson, NY.

——— AND R. E. FOX (1986), *The Race*, North River Press, Croton-on-Hudson, NY.

JAMES, S. W. AND B. A. MEDIATE, JR. (1993), *Benchmark Production Scheduling Problems For Job Shops With Interactive Constraints*, MS thesis, AFIT/GSM/LAS/93S-9, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH.

JAYSON, S. (1987), "Goldratt & Fox: Revolutionizing The Factory Floor," *Management Accounting*, 68, 11, 18–22.

LUEBBE, R. AND B. FINCH (1992), "Theory Of Constraints And Linear Programming: A Comparison," *International Journal of Production Research*, 30, 6, 1471–1478.

MORTON, T. E. AND D. W. PENTICO (1993), *Heuristic Scheduling Systems With Applications To Production Systems And Project Management*, John Wiley & Sons, Inc., New York.

PLENERT, G. (1993), "Optimizing Theory Of Constraints When Multiple Constrained Resources Exist," *European Journal of Operational Research*, 70, 1, 126–133.

RONEN, B. AND E. ROZEN (1992), "The Missing Link Between Manufacturing Strategy And Production Planning," *International Journal of Production Research*, 30, 11, 2659–2681.

——— AND M. K. STARR (1990), "Synchronized Manufacturing As In OPT: From Practice To Theory," *Computers and Industrial Engineering*, 18, 4, 585–600.

ROSE, C. (1993) Personal communication.

SCHRAGENHEIM, E. AND B. RONEN (1990), "Drum-Buffer-Rope Shop Floor Control," *Production and Inventory Management Journal*, 31, Third Quarter, 18–23.

SIMONS, J. V., JR., W. P. SIMPSON, III, B. J. CARLSON, S. W. JAMES, C. A. LETTIERE, AND B. A. MEDIATE, JR. (1996), "Formulation and Solution of the Drum-Buffer-Rope Constraint Scheduling Problem (DBRCSP)," *International Journal of Production Research*, 34, 9, 2405–2420.

SPEARMAN, M. L., D. L. WOODRUFF, AND W. J. HOPP (1990), "CONWIP: A Pull Alternative to Kanban," *International Journal of Production Research*, 28, 5, 879–894.

SPENCER, M. S. (1991), "Using *The Goal* in an MRP System," *Production and Inventory Management Journal*, 32, Fourth Quarter, 22–27.

UMBLE, M. M. AND M. L. SRIKANTH (1990), *Synchronous Manufacturing: Principles For World Class Excellence*, South-Western Publishing Co., Cincinnati, OH.