6

Open Shop Scheduling

Teofilo F. Gonzalez
University of California

6.1 Introduction

The open shop scheduling problem consists of m machines denoted by M_1, $M_2, \ldots, M_m$ that perform different tasks. There are n jobs $(J_1, J_2, \ldots, J_n)$ each of which consists of m tasks. The jth task of job J_i is denoted by $T_{i,j}$ and it must be processed by machine M_j for $p_{i,j} \geq 0$ time units. The total processing time for job J_i is $p_i = \sum_j p_{i,j}$, the processing requirement for machine M_j is $m_j = \sum_i p_{i,j}$, and we define $h = \max\{p_i, m_j\}$. The scheduling restrictions in an open shop problem are as follows:

1. Each machine may process at most one task at a time.
2. Each job may be processed by at most one machine at a time.
3. Each task $T_{i,j}$ must be processed for $p_{i,j}$ time units by machine M_j.

Clearly, the finishing time of every open shop schedule must be at least h. The main difference between the flow shop and the open shop problems is that in the former problem the tasks for each job need to be processed in order, i.e., one may not begin processing task $T_{i,j}$ until task $T_{i,j-1}$ has been completed for $1 < j \leq m$. In the open shop problem the order in which tasks are processed is immaterial. In the job shop problem jobs may have any number of tasks, rather than just m. Each task of each job is assigned to one of the machines rather than assigning the jth task to machine M_j as in the flow shop and open shop problems. But, the order in which tasks must be processed in the job shop problem is sequential as in the flow shop problem. One may think of the open shop problem as the flow shop problem with the added flexibility that the order in which tasks are processed is immaterial.

Gonzalez and Sahni [1] introduced the open shop scheduling problem back in 1974 to model several real-world applications that did not quite fit under the flow shop model. They developed a linear-time algorithm for the two machine makespan nonpreemptive as well as the preemptive scheduling problems ($O2 \,||\, C_{\max}$, and $O2 \,|\, pmtn \,|\, C_{\max}$). This result compares favorably to Johnson's two machine flow shop algorithm that takes $O(n \log n)$ time. Gonzalez and Sahni [1] also showed that for three or more machines the nonpreemptive open shop problem ($O3 \,||\, C_{\max}$) is NP-hard. Their main result was two efficient algorithms for the preemptive version of the makespan open shop problem ($O \,|\, pmtn \,|\, C_{\max}$). Since 1974 hundreds of open shop papers have been published in all sorts of conference proceedings and journals. The

minimum makespan open shop preemptive scheduling problem has found applications in many different fields of study, which is one of the reasons for the popularity of the open shop problem.

The most interesting application of the open shop preemptive scheduling problem is in scheduling theory where the problem naturally arises as a subproblem in the solution of other scheduling problems. A typical application arises when one solves (optimally or suboptimally) a scheduling problem via Linear Programming (LP). In the first step one defines a set of intervals, and a set of LP problems defines the amount of time each job is to be scheduled in each machine in each time interval. Once the set of LP programs are solved, we are left with a set of one or more open shop makespan problems. When the resulting open shop problem is such that preemptions are allowed, one can use the algorithm in Ref. [1] to construct the final schedule. Lawler and Labetoulle [2] were the first to use the open shop problem this way. Since then, it has become common practice. The most interesting use of this approach is given by Queyranne and Sviridenko [3] to generate suboptimal solutions to open shop preemptive scheduling problems with various objective functions as well as for interesting generalizations of the open shop problem.

Another application arises in the area of Satellite-Switched Time-Division Multiple Access (SS/TDMA) [4] where information has to be interchanged between multiple land sites using a multibeam satellite. The scheduling of the communications has been modeled by an open shop problem [4]. The open shop problem also arises in the scheduling and wavelength assignment (SWA) problem in optical networks that are based on the wavelength-division-multiplexing (WDM) technology [5]. Wang and Sahni [6] also use the open shop problem for routing in OTIS (optical transpose interconnect system) optoelectronic computers to find efficiently permutation routings. For mesh computers with row and column buses, the open shop problem is used for routing packets [7]. Even when routing in heterogeneous networks, the open shop problem has been used to model communications schedules [8]. Iyengar and Chakrabarty [9] used the makespan open shop preemptive scheduling problem for system-on-a-chip (SOC) testing. The computational techniques behind the makespan open shop preemptive scheduling algorithm in Ref. [1] have been applied to the solution of stochastic switching problems [10].

For the *multimessage multicasting,* Gonzalez [11–13] has developed efficient offline and online approximation algorithms not only for the fully connected networks but also for Benes-type of networks capable of replicating data and realizing all permutations. A class of approximation algorithms for the multimessage multicasting problem generate solutions by solving a couple of problems, one of which is the *multimessage unicasting* problem [12]. Gonzalez [11] has shown that the multimessage unicasting problem is equivalent to the unit-processing time makespan open shop scheduling problem [12], which can be solved by the algorithms given in Ref. [1]. The multimessage unicasting problem is also known as the h-relations problem [14] and the $(h-h)$-routing request problem.

The open shop problem is a generalization of the bipartite graph edge coloring problem. A *graph* consists of a set of V vertices and a set of edges E. A graph is said to be *bipartite* when the set of vertices can be partitioned into two sets A and B such that every edge is incident to a vertex in set A and to a vertex in set B. The *bipartite graph edge coloring problem* consists of assigning a color to each edge in the graph in such a way that no two edges incident upon the same vertex are assigned the same color and the total number of different colors utilized is least possible. The open shop problem in which all the $p_{i,j}$ values are 0 or 1 is called the *unit-processing time* open shop problem. This open shop problem with the objective function of minimizing the makespan $(O \mid p_{i,j} \in \{0,1\} \mid C_{\max})$ corresponds to the bipartite graph edge coloring problem. To see this, map the set of vertices A to the set of jobs and the set of vertices B to the set of machines. An edge from a vertex in set A to a vertex in set B represents a task with unit processing time. Each color represents a time unit. The coloring rules guarantee that an edge coloring for the graph corresponds to the unit-processing time open shop schedule. The makespan or finishing time corresponds to the number of different colors used to color the bipartite graph.

When there are multiple edges between at least one pair of nodes, the edge coloring of bipartite graphs problem is called the *bipartite multigraph edge coloring* problem. As pointed out in Ref. [4], this problem can be solved by the constructive proof of Egerváry [15], which uses König-Hall theorem [16–18]. A more general version of this problem has been defined over an edge-weighted (positive real values) bipartite

graph, and the problem is to find a set of matchings $M_1, M_2, \ldots, M_m$ and positive real-valued weights $w_1, w_2, \ldots, w_m$ such that the bipartite graph is equal to the sum of the weighted matchings. This problem corresponds to the problem solved by the Birkhoff–von Neumann theorem, which establishes that a doubly stochastic matrix (i.e., a square nonnegative real matrix with all lines (rows and columns) equal to one) is a convex combination of permutation matrices. Berge [19] presents a graph-theory based proof for this theorem and then points out: "The proof illustrates the value of the tool provided by the theory of graphs, the direct proof of the theorem of Birkhoff and von Neumann is very much longer."

The timetable problem [20,21] is a generalization of the open shop scheduling problem. The professors are the machines, the jobs are the classes, and the objective is to find times at which the professors can instruct their classes without any professor teaching more than one class at a time and any class meeting with more than one professor at a time. In addition, the classical timetable problem includes constraints where professors or classes cannot meet during certain time periods.

In Section 6.2 we discuss the open shop problem with the objective function of minimizing the makespan. We outline a linear-time algorithm for the two machine problem, as well as polynomial-time algorithms for the general preemptive version of the problem. For the nonpreemptive version of the problem we present mainly NP-hard results and approximation algorithms. We also discuss the problem of generating suboptimal solutions to these problems, as well as to the distributed version of the problem. Section 6.3 covers the open shop problem with the objective function of minimizing the mean flow time. We discuss the NP-hardness results for these problems as well as several approximation algorithms. In Section 6.4 we briefly discuss the open shop problem under various objective functions as well as generalization of the basic open shop problem.

6.2 Minimum Makespan Problems

In this section we discuss the open shop problem with the objective function of minimizing the makespan. We outline a linear-time algorithm for the two machine problem, as well as polynomial-time algorithms for the general preemptive version of the problem. For the nonpreemptive version of the problem we discuss mainly NP-hard results. We also discuss the problem of generating suboptimal solutions to these problems as well as to the distributed version of the problem.

6.2.1 Two Machines

Gonzalez and Sahni [1] developed a very clever algorithm to construct a minimum makespan schedule for the two machine nonpreemptive version of the problem $(O \,||\, C_{\max})$. This algorithm also generates an optimal preemptive schedule $(O2 \,|\, pmtn \,|\, C_{\max})$. Let us now outline a variation of this algorithm.

We represent each job by a pair of positive integers whose first component is the processing time on machine M_1 and the second component is the processing time on machine M_2 for the job. We partition these pairs into two groups: A and B. Group A contains all the tuples whose first component is greater or equal to the second, and group B contains all the tuples whose second component is larger than the first one. We will only discuss the case when both sets are nonempty, because the other cases are similar. The group A is represented as a sequence by $A_1, A_2, \ldots, A_R$ and B is represented by the sequence $B_1, B_2, \ldots, B_L$. The processing time on machine M_j for tuple A_i is denoted by $A_i(j)$. Similarly, we define $B_i(j)$. From our definitions we know that $A_i(1) \geq A_i(2)$, and $B_j(1) < B_j(2)$. We assume without loss of generality that A_R is the job in A with largest processing time on machine M_1, i.e., $A_R(1)$ is largest. Similarly, B_L is the job in B with largest processing time on machine M_2.

Now construct the schedule that processes jobs in the order $A_1, A_2, \ldots, A_R$ as shown in Figure 6.1(b). For each job the task on machine M_1 is scheduled immediately after the completion of the previous task (if any) on machine M_1, and the task on machine M_2 is processed at the earliest possible time after the completion of the job's task on machine M_1. Clearly, the only idle time between tasks is on machine M_2. Since $A_R(1)$ is the largest value, then $A_R(1) \geq A_{R-1}(2)$. Therefore, the task on machine M_2 for job A_R

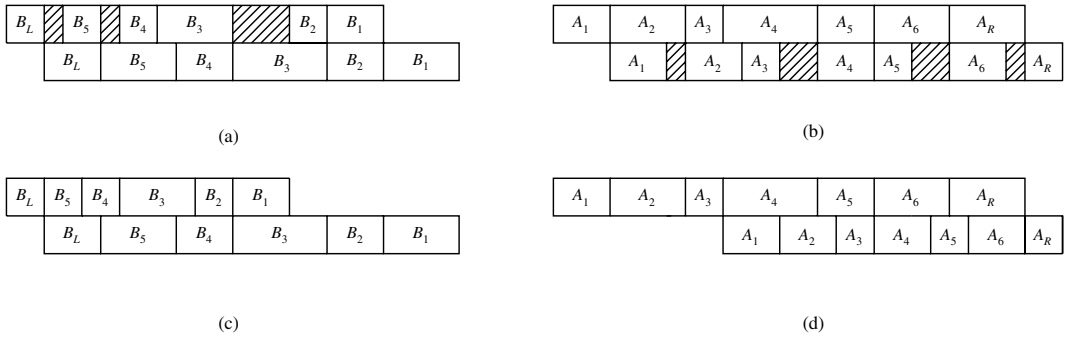

FIGURE 6.1 Schedule for the jobs in A and B.

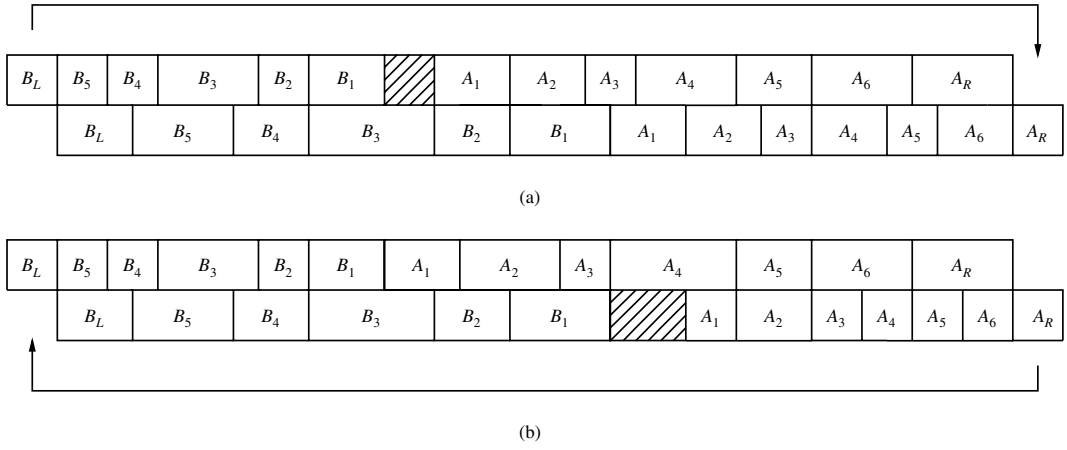

FIGURE 6.2 Schedule after joining the schedules for A and B.

starts at the same time as its task on machine M_1 ends. Since every job in A is scheduled so that its task on machine M_1 is completed before the task on machine M_2 begins, we can delay the processing of the tasks scheduled on machine M_2 (i.e., move them to the right) until we eliminate all the idle time between tasks and the schedule of job A_R is not changed (see Figure 6.1(d)).

We obtain a similar schedule for B (see Figure 6.1(a)). Note that in this case the idle time between tasks is only on machine M_1, and the tasks are processed in the order $B_1, B_2, \ldots, B_L$ but from right to left. In this case the tasks scheduled on machine M_1 are moved to the left to eliminate idle time between tasks (see Figure 6.1(c)).

Now we concatenate the schedule for B and A in such a way that there is no idle time between the schedules on machine M_1 (Figure 6.2(a)), or machine M_2 (Figure 6.2(b)), or both (either of the figures without idle time between the schedules on both machines).

If one ends up with the schedule in Figure 6.2(a), we move job B_L on machine M_1 from the front of the schedule to the back. We then push all the tasks to the left until the block of idle time between the schedules is eliminated (Figure 6.3(a) or 6.3(b)), or until job B_L starts on machine M_1 at the time when it finished on machine M_2 (Figure 6.3(c)). In the former case the total makespan is equal to $m_1 = \sum A_i(1) + \sum B_j(1)$ or $m_2 = \sum A_i(2) + \sum B_j(2)$, and in the latter case it is given by $B_L(1) + B_L(2)$. On the other hand, if we end up with the schedule in Figure 6.2(b), a similar operation is performed, but now we move job A_R on machine M_2. The resulting schedules are similar to those in Figure 6.3 except that job A_R replaces job B_L and the machines are interchanged. Clearly the time complexity of the algorithm is $O(n)$.

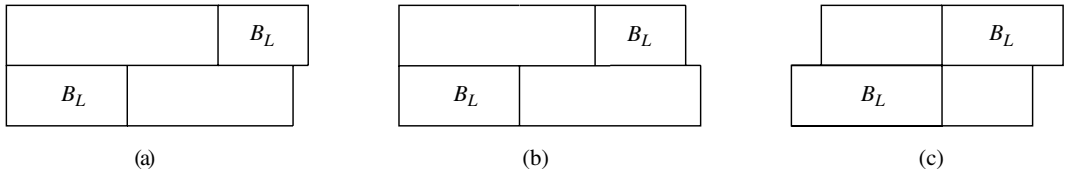

FIGURE 6.3 Schedule after moving task B_L on machine M_1.

Since the finishing time for the schedule constructed is $\max\{m_1, m_2, A_R(1) + A_R(2), B_L(1) + B_L(2)\}$, which is simply h, there are no preemptions in the schedule, and every schedule must have finishing time greater than or equal to h, it then follows that the schedule constructed by our procedure is a minimum makespan nonpreemptive preemptive schedule, i.e., it solves the $O \,||\, C_{\max}$ and $O \mid pmtn \mid C_{\max}$ problems.

6.2.2 Minimum Preemptive Schedules

Now we outline the polynomial-time algorithms developed by Gonzalez and Sahni [1] for the $O \mid pmtn \mid C_{\max}$ problem. The first step in both of these algorithms consists of transforming the problem, by introducing dummy machines and jobs, to one in which for all i and j, $p_i = h$ and $m_j = h$. Since this first operation is straightforward, we will omit it and just assume that the above condition holds. We use r to denote the number of nonzero $p_{i,j}$ values, which we shall refer to as *nonzero* tasks.

The first algorithm in Ref. [1] begins by constructing an edge-weighted bipartite graph G in which there is one vertex for each job J_i and one vertex for each machine M_j. All the edges join a vertex that represents a job and a vertex that represents a machine. For every nonzero task $T_{i,j}$ (i.e., $p_{i,j} > 0$) there is an edge with weight $p_{i,j}$ from the vertex representing job J_i to the vertex representing M_j.

A *matching* for G is a set of edges no two of which have a common end point. A *complete matching* is a matching in which every vertex in the graph is adjacent to an edge in the matching. Using Hall's theorem one can show that G has a complete matching M [1]. Define Δ as the smallest weight of an edge in the matching M. The matching M defines for the first Δ time units the assignment of tasks to machines. Then in G we decrease by Δ the weight of every edge in the matching and obtain the new graph G. When the weight of an edge becomes zero, the edge is deleted from the graph. The whole process is repeated until there are no edges left in the graph.

It should be clear that at each iteration at least one edge is deleted from the graph. Therefore there are at most r iterations. Using Hopcroft and Karp's algorithm [22] the first complete matching can be constructed in $r(n+m)^{0.5}$. To find subsequent matchings one resorts to using the previous matching M, after deleting all the edges with weight Δ. For each of the edges that were deleted one needs to find one augmenting path. An *augmenting (or alternating) path* relative to a matching M is a simple path with an odd number of edges such that the ith edge in the path, for i odd, is in the graph but not in the matching, and for i even, it is an edge in the matching. The "even" edges are represented by the set M' and the "odd" ones are represented by the set of edges M_N. If we delete from M all the edges in M' and then we add all the edges in M_N, we obtain a new matching that contains one more edge than the matching we had before the operation. The reader is referred to Gonzalez and Sahni [1] for the precise definitions and procedures to find augmenting paths. The time required to find an augmenting path is $O(r)$ and it is obtained via breadth first search. So, if the complete matching M had l edges with weight Δ, then one needs to construct l augmenting paths to find the next complete matching. Therefore this technique can be implemented to take $O(r^2)$ time. The above approach is essentially the typical constructive proof for the Birkoff–von Neumann theorem [18] and Egerváry theorem [15]. As pointed out by Berge [19], the proof of the Birkoff–von Neumann theorem when viewing the problem as a graph problem is much simpler than the original one.

The second algorithm given by Gonzalez and Sahni [1] takes $O(r(\min\{r, m^2\} + m \log n))$ time. The algorithm is designed for the case when $m < n$, and it is better than the first one when $r > m^2$. As in the first algorithm, the first step adds dummy machines and jobs so that $p_i = m_j = h$ for all i and j. Then all

the dummy jobs are deleted. As a result of this operation some jobs will have total processing time less than h. The jobs with $p_i = h$ are called *critical* and those with $p_i < h$ are called *noncritical*. All the machines are called *critical* because $m_j = h$ for all j. The strategy behind the algorithm is similar to that in the previous algorithm. The main difference is that the cost of an augmenting path is m^2 rather than r. In the first step we find a matching I that includes all critical jobs and machines. We build this matching by finding and then using an augmenting path for each critical job, and then finding and using an augmenting path for each critical machine that is not in the matching. We define the slack time s_i of a job as $h - c - g_i$, where c is the current time (this is just the makespan of the schedule so far constructed), and g_i is the remaining processing time for the job. Initially $g_i = p_i$ and $c = 0$. A noncritical job becomes critical when its slack time s_i becomes zero. Remember that at every point in time t all jobs that are critical at time t must be scheduled, i.e., must be in the matching I at time t. Therefore the minimum slack time for the jobs that are not in the matching at time t is greater than zero. We define Δ as the minimum of the remaining processing time of a task represented by an edge in the matching I and the minimum slack time of the jobs that are not in the matching. The minimum slack time can be computed in $O(\log n)$ time by storing all the slack times of the jobs that are not in the matching in a balanced tree. Then we generate the schedule implied by the matching for the next Δ time units. We update the slack times for the noncritical jobs in the matching, which takes $O(m)$ time. We delete all the noncritical jobs from the matching, and delete all the tasks with remaining processing time zero. Then we add to the resulting matching any critical jobs that are not in the matching. Each of these additions are carried out by finding and then using an augmenting path that takes $O(m^2)$ time to construct, as shown in Ref. [1]. Now we add to the matching any of the noncritical jobs that were just deleted. For these jobs we do not find an augmenting path, we just add them if the corresponding machine for the task that was part of the previous matching is not covered by the current matching. Then through the augmenting path technique we add all the critical machines that are not in the matching and we end up with another matching that includes all the critical jobs and critical machines. The whole process is repeated until all the tasks have been scheduled for the appropriate amount of time. Readers interested in the complete details (which are complex) are referred to Gonzalez and Sahni [1]. The total number of iterations (matchings) constructed is at most $r + m$ because at each iteration at least one nonzero task has been scheduled for its full processing time, or a noncritical job becomes critical. Since $r \geq m$ and the cost of each augmenting path is at most m^2, all of these operations take $O(r \min\{r, m^2\})$ time. The other factor in the time complexity bound, $r \cdot m \log n$, originates for the operation of updating the slack time in the balanced binary search tree of at most m tasks at each iteration. This time also includes the time to find the smallest slack time of a task that is not in the matching. Therefore, the overall time complexity bound for the second algorithm is $O(r(\min\{r, m^2\} + m \log n))$. The total number of preemptions introduced by the algorithm is at most $r(m - 1) + m^2$ preemptions.

The algorithm for the minimum makespan open shop preemptive scheduling problem by Vairaktarakis and Sahni [23] has the same worst case time complexity; however, it is in general faster and requires less space than the previous two algorithms. This is achieved by not introducing dummy jobs and dummy machines, and only critical jobs and machines are required to be in the matching at each step. This eliminates the overhead of dealing with the additional jobs and machines. The added constraints of *load balancing* as well as *maintenance* are incorporated into the algorithms given in Ref. [23]. The former constraints balance the number of busy machines throughout the schedule, and the latter constraints deal with machine maintenance with limited personnel. The algorithms in Ref. [23] use linear programming, max flow as well as additional graph theory properties of bipartite matchings.

The algorithms in Refs. [1,23] are in general much more efficient than the algorithms that were developed for the corresponding bipartite graph and multigraph edge coloring problems as well as for the doubly stochastic matrices.

As we mentioned before, Gonzalez [11] has established the equivalence of the makespan open shop preemptive scheduling problem and the multimessage unicasting problem. In this problem each processor must send equal size messages to other processors over a fully connected network. The scheduling rules are that no processor may send more than one message at a time and no processor may receive more than one message at a time. The objective is to find a schedule with the least total completion time.

In the distributed version of the multimessage unicasting problem every processor only knows the messages it will be sending and does not know what other processors have to do. Though every processor may send or receive at most d messages. When this is translated back into the open shop problem we need to introduce n users each of which is trying to process its corresponding job on the machines without knowing what the other users are trying to do. At each time unit each user decides which (if any) of its task he/she will attempt to get processed by the corresponding machine. If more than one user attempts to use the same machine, then the machine will be idle and both users will be informed that their tasks were not processed at that time. Gereb-Graus and Tsantilas [24] presented distributed algorithms with $\Theta(d + \log n \log\log n)$ expected communication steps. The multimessage unicasting and multicasting problems with forwarding have been studied in the context of optical-communication parallel computers [14,24–26]. Forwarding means that a message does not need to be sent directly, the message may be sent through another machine. In Section 6.4 we explain what forwarding means in the context of open shop scheduling.

6.2.3 Limiting the Number of Machines, Jobs or Tasks

First let us consider the open shop preemptive scheduling problem when either the number of jobs or machines is not part of the problem input, e.g., the number of machines is at most 20 (or any other constant) but the number of jobs may be any arbitrary number, or vice versa. Gonzalez [27] shows that these problems can be solved in linear time by using a technique called *combine-and-conquer*. The main idea is that any subset of jobs or machines whose total processing time is at most h can be combined into a super-job or a super-machine. From a solution to the super-problem one can easily solve the original problem. Obviously this combine-and-conquer approach is not recursive, it is applied just once. The selection of which jobs to combine is determined by solving an instance of the bin-packing problem. The *bin-packing* problem consists of packing into the least number of bins with capacity h a set of objects whose size corresponds to the job processing times $p_1, p_2, \ldots, p_1$. If $k = \min\{n, m\}$, then $\sum p_i \leq k \cdot h$. There are simple linear-time algorithms that pack all of these objects in at most $2k - 1$ bins each of size h. The same approach is used for the machines. Therefore we will end up with a problem instance that has at most $2k - 1$ (super) jobs and $2k - 1$ (super) machines. Since there is a fixed number of jobs or machines (independent of the input), the resulting problem has a number of jobs and machines that is bounded by a constant independent of the input. Any of the algorithms in the previous subsection can be used to solve this reduced size problem and the solution can be easily used to obtain a solution to the original problem. The above procedure can be easily implemented to take $O(n + m)$ time.

Let us now discuss algorithms that perform very well when every job has very few tasks, and every machine needs to process very few tasks. From our previous discussion this problem may be viewed as the multigraph edge coloring problem. Gabow and Kariv's [28] algorithm to color the edges of a bipartite multigraph takes time $O(\min\{m \log^2 n, n^2 \log n\})$, where n is the number of nodes in the graph and m is the number of edges. Cole and Hopcroft [29] developed a faster algorithm for the same problem with time complexity bounded by $O(m \log n)$. Cole and Hopcroft's [29] algorithm uses the combine-and-conquer approach [27] as well as the idea of finding a matching with only critical jobs as in the algorithms in the previous subsection [1]. These algorithms are the fastest ones when the degree of the multigraph is small. When the edge multiplicity is large, it is better to use Gonzalez and Sahni's [1] open shop algorithms, because multiple edges are treated as a single weighted edge and the schedule for a whole interval may be generated at each iteration, rather than one for one time unit.

6.2.4 Nonpreemptive Schedules

To establish that the minimum makespan open shop nonpreemptive scheduling problem is NP-hard, Gonzalez and Sahni [1] reduced the partition problem to three machine problem instances ($O3 \,||\, C_{\max}$). Given n objects denoted by $a_1, a_2, \ldots, a_n$ and a size or weight function $s\colon a \rightarrow I^+$ the *partition* problem is to determine whether or not the set A can be partitioned into two sets, A_1 and A_2, such that the sum of the weight (or size) of the objects in each set is equal to $T/2$, where $T = \sum s(a_i)$.

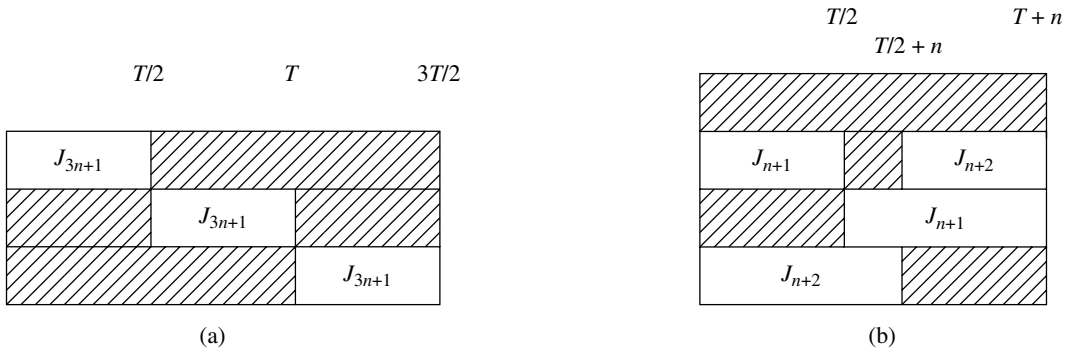

FIGURE 6.4 Architecture of reductions from partition.

The reduction to the three machine nonpreemptive open shop makespan problem ($O3 \,||\, C_{\max}$) constructs an instance from partition with $3n + 1$ jobs and three machines. The last job, J_{3n+1}, has processing time on each machine of $T/2$ time units. Therefore, if there exists a schedule with makespan equal to $3T/2$, then job J_{3n+1} has to be processed without interruptions. Since the schedule cannot have preemptions, it must be that on one of the machines the task from J_{3n+1} is processed from time $T/2$ to time T (see Figure 6.4(a)). That leaves two disjoint blocks of idle time each of length $T/2$ on one machine. These blocks will be used to process a set of n tasks whose processing time corresponds to the size of the objects in the instance of partition. Therefore, a schedule with finishing time at most $3T/2$ exists iff the instance of partition we start from has a partition for set A. To make sure that such a set of tasks exists, we need to introduce $3n$ jobs as follows. For $1 \leq j \leq 3$, the jth set of n jobs have only nonzero tasks on machine M_j and their processing time for the ith job on machine M_j is $s(a_i)$. Therefore, the open shop makespan decision problem is NP-hard even when $m = 3$ [1].

All the jobs in the above construction have one nonzero task except for the last job that has nonzero processing time on the three machines. Does the problem remain NP-hard even when each job has at most two nonzero tasks? Gonzalez and Sahni [1] addressed this problem and showed that it remains NP-hard when $m \geq 4$. The reduction is similar in nature to the previous one. The difference is that now there are two jobs with processing requirements such that in every schedule with makespan at most $T + n$, they leave a block of idle time from time $T/2$ to $T/2 + n$ on machine M_2, and M_1 is not utilized (see Figure 6.4(b)). A set on n jobs is introduced each with processing time of 1 unit on machine M_2 and a processing time corresponding to the size of an object in the partition problem on machine M_1. By scaling up the size of the objects we can guarantee that if there is partition with total size between $T/2$ and $T/2 + n$, then there is also a partition of size $T/2$. This will guarantee that a schedule with finishing time at most $T + n$ exists iff we start with a yes-instance of partition. Therefore, the minimum makespan open shop scheduling problem is NP-hard even when every job has at most two nonzero tasks and $m = 4$ [1].

The above reductions do not establish that the open shop problem is NP-hard in the strong sense [30], i.e., the problem is not shown to be NP-hard when the sum of the task times is bounded by a polynomial of n and m. This is because the partition problem is not NP-complete in the strong sense unless $P = \text{NP}$. However, Lenstra [31,32] has shown that the open shop problem is NP-hard in the strong sense for an arbitrary number of machines. To show that this problem is NP-hard in the strong sense we reduce the 3-partition problem to it. In the 3-partition problem we are given m objects and a size or weight function defined as in the partition problem. The problem is to decide if the set of objects can be partitioned in $m/3$ subsets such that the sum of the size of the objects in each subset is identical. Figure 6.5 gives the architecture of the reduction for the case when $m = 18$. The main idea is to introduce a set of jobs that no matter how they are assigned in a schedule with certain finishing time, there will be $m/3$ equally sized blocks of idle time on machine M_1 whose total size corresponds to the sum of the size of the objects

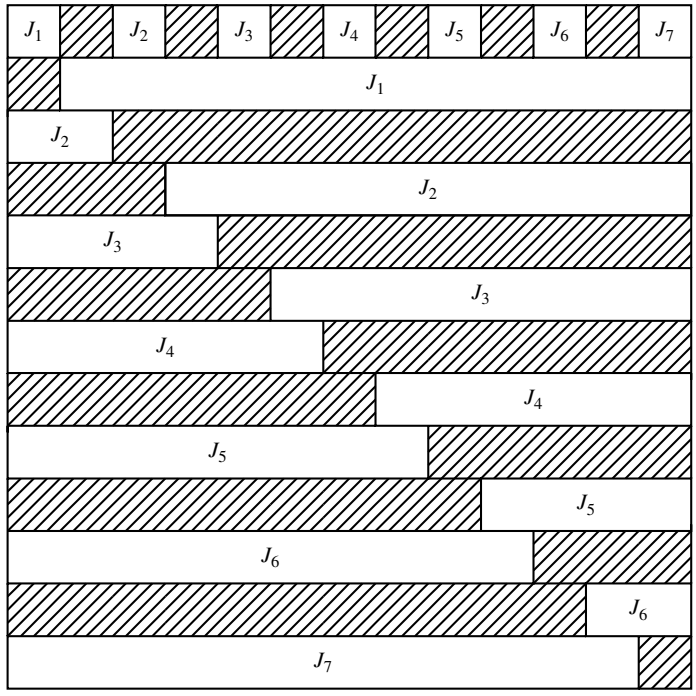

FIGURE 6.5 Architecture of reduction from 3-partition.

in the instance of 3-partition we start from. Without further discussion of the details we claim that the minimum makespan open shop nonpreemptive scheduling problem ($O \,||\, C_{\max}$) is NP-hard in the strong sense [31].

Another interesting restricted open shop problem is one in which all the nonzero tasks have the same processing time (or unit processing time, $O \mid p_{i,j} \in \{0, 1\} \mid C_{\max}$). For this case it is simple to see that the preemptive scheduling algorithms presented in the previous section generates a minimum makespan schedule.

There is a very simple approximation algorithm for the makespan open shop nonpreemptive scheduling problem ($O \,||\, C_{\max}$). This algorithm was developed and analyzed by Racsmány (see [33]), but a better approximation bound for the algorithm is given by Shmoys, Stein, and Wein [33]. The scheduling strategy mimics Graham's well-known list scheduling algorithm. Whenever a machine becomes available we assign to each available machine a task which has not yet been processed on that machine that belongs to a job that is not currently being processed by another machine. This very simple procedure generates a schedule with finishing time, which is at most twice the length of an optimal schedule. To see this, consider a job that finishes at the latest time in the schedule just constructed. Let us say it is job J_i and the latest time it is being processed is on machine M_j. The finishing time on machine M_j (which is the finishing time of the schedule) is equal to the processing time demands on that machine, which we have previously defined to be at most h plus the total idle time on machine M_j. Now, when there is idle time on machine M_j it must be that a task of job J_i was being processed by another machine, as otherwise the scheduling algorithm should have assigned the task of job J_i to machine M_j. Therefore the total idle time on machine M_j is less than the total processing time for job J_i, which is equal to p_i. Since by definition $p_i \leq h$, it then follows that the algorithm generates a schedule with finishing time at most $2h$. Clearly, every schedule must have finishing time at least h. So it follows that the schedule generated by the above procedure has makespan at most two times the optimal makespan.

Williamson et al. [34] showed that the problem of generating schedules within 5/4 times the length of the optimal schedule for the open shop nonpreemptive scheduling problem is NP-hard. They also present

a polynomial-time algorithm for the case when for all i and j, $p_{i,j} \in \{0, 1, 2, 3\}$, and there is an optimal schedule with finishing time at most 3.

6.3 Minimum Mean Flow Time or Minsum Problems

In this section we discuss the open shop problem with the objective function of minimizing the mean flow time. Since the minimum mean flow time, $\sum C_i / n$, objective function is equivalent to the minsum one, $\sum C_i$, we use these terms interchangeably. We discuss the NP-hardness results for these problems as well as several approximation algorithms.

Achugbue and Chin [35] have established that the minsum problem is NP-complete in the strong sense even for two machines ($O2 \,||\, \sum C_i$). Their reduction is quite complex and it is similar in architecture to that for the flow shop by Garey, Johnson, and Sethi [36]. Liu and Bulfin [37] showed that the problem is NP-hard in the strong sense for three machines when preemptions are allowed by using a reduction from 3-partition ($O3 \mid pmtn \mid \sum C_i$). Subsequently Du and Leung [38] showed that the minsum open shop preemptive scheduling problem is NP-hard (in the normal sense) even when there are only two machines ($O2 \mid pmtn \mid \sum C_i$) using a reduction from a restricted version of partition.

The simplest version of the minsum open shop problem is when all the nonzero tasks have equal processing times ($O \mid p_{i,j} \in \{0, 1\} \mid \sum C_i$ and $O \mid p_{i,j} \in \{0, 1\}; pmtn \mid \sum C_i$). Gonzalez [39] showed that this problem is NP-hard in both the preemptive and nonpreemptive mode. Since the flavor of this reduction is quite different from most other reductions, we explore it in more detail. The reduction is from the graph coloring problem. The *graph coloring* problem is given an undirected graph assign a color to each vertex in the graph in such a way that the least number of colors is used and no two adjacent vertices are assigned the same color. The reduction from the graph coloring problem to the open shop problem is as follows. The set of jobs represents nodes, edges, and node-edge pairs in the graph, and the machines represent nodes and edges in the graph. Time is partitioned into three different intervals, each corresponds to one color. The node jobs force the corresponding node-edge job to be confined to one of the three time intervals. The edge jobs are introduced to simplify the accounting of the objective function value. The node-edge jobs are defined in such a way that two jobs that represent adjacent nodes in the graph must be scheduled in different time intervals. The jobs and machines are defined in such a way that if the graph we start from is three colorable, then one-third of the jobs finish at time 5, another third finish at time 10, and the remaining third finish at time 15, with a total mean flow time of 10. When the graph is not three colorable, then all schedules have mean flow time greater than 10. The whole reduction is quite complex, so readers interested in additional details are referred to Ref. [39]. The reduction does not work for the makespan problem ($O \mid p_{i,j} \in \{0, 1\} \mid C_{\max}$) because for graphs that are not three colorable there are schedules with finishing time equal to 15, though it will not be the case that one-third of the jobs finish at time 5 and the next third of the jobs finish at time 10. But the reduction does work for the minimum makespan open ship no-wait scheduling problem. By *no-wait* scheduling we mean that all the tasks from a job must be executed contiguously.

Achugbue and Chin [35] showed that any schedule that does not have idle time on all the machines at the same time for the $O \,||\, \sum C_j$ problem has total completion time that is at most n times the optimal one. They also showed [35] that the simple SPT scheduling rule guarantees that the total completion time of the schedules generated is no more than m times the optimal one. Hoogeveen, Schuurman, and Woeginger [40] showed that generating near optimal solutions for the $O \,||\, \sum C_j$ problem is as difficult (computationally) as generating an optimal solution. The approximation bound for which the problem is NP-hard is somewhere in the $1 + 10^{-5}$ range. They showed that this problem is APX-complete, which means that if the problem has a polynomial-time approximation scheme, then $P = \text{NP}$.

Queyranne and Sviridenko [3] developed a quite sophisticated approximation algorithm for the open shop preemptive problem (as well as for more general versions of the problem) under different objective functions. The approximation bound is $(2+\epsilon)$. The idea is to have an interval-indexed formulation, which may be viewed as a set of intervals defined in terms of ϵ and another small constant δ over which an LP

problem is defined. From the solution to the LP problem they construct a schedule using the algorithm for $O \mid pmtn \mid C_{\max}$ given in Ref. [1]. Then randomization is used through Schultz and Skutella's slow-motion algorithm with the factor β being randomly chosen with certain properties. After this process there is a derandomization step. The analysis of this process is quite complex. They showed that this technique cannot generate solutions with an approximation factor better than 2. However, the algorithm works for generalized versions of the open shop. For example, when a task may be processed on several machines at different speeds, or the objective function includes the finishing time of tasks rather than just the finishing time of the jobs.

6.4 Related Objective Functions

In this section we discuss briefly the open shop problem under various objective functions as well as generalization of the basic open shop problem.

The open shop problem has been studied under other classical objective functions as well as other restrictions. All of these results are very important, but for brevity we cannot possibly discuss all of them. Chapter 9 and Chapter 10 discuss algorithms for other objective functions, and there are several lists of current results for open shop problems available on the Internet (e.g., Ref. [41]). We will just point to some polynomial-time algorithms that have received attention in the past: $O \mid pmtn; r_i \mid L_{\max}$ by Cho and Sahni [42], $O \mid p_{ij} = 1; intree \mid L_{\max}$ by Brucker [43], $O2 \mid p_{ij} = 1; prec \mid \sum C_i$ by Coffman et al. [44], and $O \mid p_{ij} = 1 \mid \sum T_i$ by Liu and Bulfin [37]. Recent NP-hard results for restricted open shop problems are given by Timkovsky [45].

Vairaktarakis and Sahni [23] also present algorithms for some very interesting extensions of the open shop problem. One of these problems is called the *generalized open shop* problem that allows multiple copies of the same machine, but the same scheduling constraints remain. They also define the *flexible open shop*, where a machine is allowed to perform different tasks, not just one as in the open shop. The algorithms in Ref. [23] use linear programming, max flow, as well as additional graph theory properties of bipartite matchings.

The multimessage multicasting problem may be viewed as a more general open shop problem. In this generalization, each task of a job consists of a subset of subtasks each of which is to be processed by a different machine. But the processing of the subtasks of a task may be concurrent. This problem has been shown to be NP-hard even when all subtasks have unit-processing time [11]. However, Gonzalez [11,12] has developed efficient approximation algorithms for this problem. The most efficient ones use message forwarding. That means that a message that needs to be sent from processor i to processor j may be sent indirectly. For example, first it may be sent from processor i to processor l, and then from processor l to processor j. When we translate forwarding to the multimessage unicasting problem, which is equivalent to the open shop problem, we obtain another new version of the open shop problem that allows for solutions whose scheduling has an added flexibility. The added flexibility is that if job J_i needs to be processed by machine M_j, it can be replaced by job J_i that needs to be processed by machine M_l and then job J_l needs to be processed by machine M_j, provided that the second job is performed after the first one. Another important point is that the resulting open shop that needs to be solved is such that some tasks of some jobs may be processed concurrently when forwarding is allowed. This added scheduling flexibility simplifies the scheduling problem, except when each message is to be delivered to just one processor.

In the distributed version of the multimessage multicasting problem every processor only knows the messages it must send. Gonzalez [13] has developed approximation algorithms for this problem, which reduce the problem to the solution of two problems, one of which is the multimessage unicasting problem with forwarding, and in the previous paragraph we explained how forwarding is translated into the open shop problem. In a previous section we discussed the meaning of the distributed version of the problem in the context of open shop scheduling.

The distributed version of the multimessage unicasting problem with forwarding (which corresponds to a form of open shop problem) has been studied in the context of optical-communication parallel

computers [14,24–26]. Valiant [26] presented a distributed algorithm with $O(d + \log n)$ total expected communication cost. The algorithm is based in part on the algorithm by Anderson and Miller [25]. The communication time is optimal, within a constant factor, when $d = \Omega(\log n)$, and Gereb-Graus and Tsantilas [24] raised the question as to whether a faster algorithm for $d = o(\log n)$ exists. This question was answered in part by Goldberg, Jerrum, Leighton, and Rao [14] who show all communication can take place in $O(d + \log\log n)$ communication steps with high probability, i.e., if $d < \log n$, then the failure probability can be made as small as n^{α} for any constant α.

6.5 Discussion

The decision version of all the NP-hard scheduling problems that were discussed in this chapter can be shown to be NP-complete. For the preemptive scheduling problems one needs to establish the minimum number of preemptions needed by an optimal solution.

Even though the open shop problem is relatively young, there have been several hundred papers dealing with this problem. The main popularity of the problem is that it models a large number of real-world problems. The algorithms are quite interesting and many of NP-hard reductions are quite complex. The main invariant of all of the work is that as we explore more of the open shop problem we find more interesting versions, generalizations, as well as applications of the problem.

References

[1] Gonzalez, T.F. and Sahni, S., Open shop scheduling to minimize finish time, *J. ACM,* 23, 665, 1976.

[2] Lawler, E.L. and Labetoulle, J., On preemptive scheduling of unrelated parallel processors by linear programming, *J. ACM,* 25, 612, 1978.

[3] Queyranne, M. and Sviridenko, M., A $(2+\epsilon)$-approximation algorithm for generalized preemptive open shop problem with minsum criteria, *J. Algor.,* 45, 202, 2002.

[4] Dell'Amico, M. and Martello, S., Open shop, satellite communication and a Theorem by Egerváry, *Oper. Res. Lett.,* 18, 207, 1996.

[5] Bampis, E. and Rouskas, G.N., The scheduling and wavelength assignment problem in optical WDM networks, *IEEE/OSA J. Lightwave Technol.,* 20, 782, 2002.

[6] Wang, C.F. and Sahni, S., OTIS optoelectronic computers, in Li, K. and Zheng, S.Q. (eds.), *Parallel Computing Using Optical Interconnections,* Kluwer, 1998, pp. 99–116.

[7] Suel, T., Permutation routing and sorting on meshes with row and column buses, *Parallel Proc. Lett.,* 5, 63, 1995.

[8] Bhat, P.B., Prasanna, V.K., and Raghavendra, C.S., Block-cyclic redistribution over heterogeneous networks, *Cluster Comp.,* 3, 25, 2000.

[9] Iyengar, V. and Chakrabarty, K., System-on-a-chip test scheduling with precedence relationships, preemption, and power constraints, *IEEE CAD,* 21, 1088, 2002.

[10] Altman, E., Liu, Z., and Righter, R., Scheduling of an input-queued switch to achieve maximal throughput, *Probab. Eng. Inf. Sci.,* 14, 327, 2000.

[11] Gonzalez, T.F., Complexity and approximations for multimessage multicasting, *J. Par. Dist. Comput.,* 55, 215, 1998.

[12] Gonzalez, T.F., Simple multimessage multicasting approximation algorithms with forwarding, *Algorithmica*, 29, 511, 2001.

[13] Gonzalez, T.F., Distributed multimessage multicasting, *J. Interconnection Networks,* 1, 303, 2000.

[14] Goldberg, L.A., Jerrum, M., Leighton, F.T., and Rao, S., Doubly logarithmic communication algorithms for optical-communication parallel computers, *SIAM J. Comput.*, 26, 1100, 1997.

[15] Egerváry, E., Matrixok kombinatorius tulajdonságairol, *Matematikai és Fizikai Lapok,* 38, 16, 1931. (English translation by Kuhn, H.W., On combinatorial properties of matrices, *Logistic Papers,* 11, 1, 1955, George Washington University.)

[16] König, D., Graphos és matrixok, *Matematikai és Fizikai Lapok,* 38, 116, 1931.
[17] Hall, M., *Combinatorial Theory,* Blaisdell, Waltham, MA, 1967.
[18] Birkoff, G., Tres observaciones sobre el algebra lineal, Revista Facultad de Ciencias Exactas, Puras y Aplicadas, Universidad Nacional de Tucuman, Series A (Mathematicas y Ciencias Teoricas), 5, 147, 1946.
[19] Berge, C., *The Theory of Graphs and it Applications,* Wiley, 1962.
[20] Gotlieb, C.C., The construction of class-teacher timetables, *Proc. IFIP Congress*, 1962, pp. 73–77.
[21] Even, S., Itai, A., and Shamir, A., On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.*, 5, 691, 1976.
[22] Hopcroft, J. and Karp, R.M., An $n^{2.5}$ algorithm for maximum matchings in bipartite graphs, *SIAM J. Comput.,* 2, 225, 1973.
[23] Vairaktarakis, G. and Sahni, S., Dual criteria preemptive open shop problems with minimum finish time, *Naval Res. Logistics,* 42, 103, 1995.
[24] Gereb-Graus, M. and Tsantilas, T., Efficient optical communication in parallel computers, *Proc. 4th ACM Symp. Parallel Algor. Architect.,* ACM, New York, Vol. 41, 1992.
[25] Anderson, R.J. and Miller, G.L., Optical communications for pointer based algorithms, TRCS CRI 88 – 14, USC, Los Angeles, 1988.
[26] Valiant, L.G., General purpose parallel architectures, in van Leeuwen, J. (ed.), *Handbook of Theoretical Computer Science,* Elsevier, New York, Chap. 18, 1990, p. 967.
[27] Gonzalez, T.F., A note on open shop preemptive schedules, *IEEE Trans. Comput.,* 28, 782, 1979.
[28] Gabow, H., and Kariv, O., Algorithms for edge coloring bipartite graphs and multigraphs, *SIAM J. Comput.*, 11, 117, 1982.
[29] Cole, R. and Hopcroft, J., On edge coloring bipartite graphs, *SIAM J. Comput.,* 11, 540, 1982.
[30] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.
[31] Lenstra, J.K. (unpublished).
[32] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., and Shmoys, D.B., Sequencing and scheduling: algorithms and complexity, in Graves, S.C., Rinnooy Kan, A.H.G., and Zipkin, P.H. (eds.), *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory,* North-Holland, 1993.
[33] Shmoys, D.B., Stein, C., and Wein, J., Improved approximation algorithms for shop scheduling problems, *SIAM J. Comput.,* 23, 617, 1994.
[34] Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevastianov, S.V., and Shmoys, D.B., Short shop schedules, *Oper. Res.,* 45, 288, 1997.
[35] Achugbue, J.O. and Chin, F.Y., Scheduling the open shop to minimize mean flow time, *SIAM J. Comput.,* 11, 709, 1982.
[36] Garey, M.R., Johnson, D.S., and Sethi, R., The complexity of flowshop and jobshop scheduling, *Math. Oper. Res.,* 1, 117, 1976.
[37] Liu, C.Y. and Bulfin, R.L., On the complexity of preemptive open shop scheduling problems, *Oper. Res. Lett.,* 4, 71, 1985.
[38] Du, J. and Leung, J.Y.-T., Minimizing mean flow time in two-machine open shops and flow shops, *J. Algor.,* 14, 24, 1993.
[39] Gonzalez, T.F., Unit execution time shop problems, *Math. Oper. Res.,* 7, 57, 1982.
[40] Hoogeveen, H., Schuurman, P., and Woeginger, G.J., Nonapproximability results for scheduling problems with minsum criteria, *INFORMS J. Comput.*, 13, 157, 2001.
[41] Brucker, P., Hurink, J., and Jurisch, J., Operations Research: complexity results of scheduling problems, www.mathematik.uni-osnabrueck.de/research/OR/class/.
[42] Cho, Y. and Sahni, S., Preemptive scheduling of independent jobs with release and due times on open, flow and job shops, *Oper. Res.,* 29, 511, 1981.
[43] Brucker, P., Jurisch, B., and Jurisch, M., Open shop problems with unit time operations, *Z. Oper. Res.,* 37, 59, 1993.

[44] Coffman, Jr., E.G. and Timkovsky, V.G., Ideal two-machine schedules of jobs with unit-execution-time operations, *Proc. 8th Inter. Workshop on Proj. Manag. Sched.,* Valencia, Spain, April 2002.
[45] Timkovsky, V.G., Identical parallel machines vs. unit-time shops, preemptions vs. chains, and other offsets in scheduling complexity, Technical Report, Star Data Systems, Inc. 1998.
[46] Sahni, S. and Gonzalez, T.F., P-complete approximation problems, *J. Assoc. Comput. Machinery*, 23, 555, 1976.