Exploiting the Scheduling Algorithms in SAS/OR® Software

Radhika Kulkarni, SAS Institute Inc., Cary, NC

Abstract

The scheduling algorithms that are part of the SAS/OR software product, when combined with the flexible programming and graphical tools available in the SAS® System, provide a rich environment for modeling problems that arise in production and operations management. The main purpose of this paper is to show that the use of the CPM procedure is not limited to traditional project management situations. Several heuristic algorithms used in job shop or flow shop scheduling can be easily programmed using PROC CPM, as shown in this paper. In addition, some of the new features in PROC CPM that allow you to associate calendars with resources and specify resource driven durations are illustrated in a manpower scheduling example.

Introduction

The CPM procedure in SAS/OR software is designed to schedule the activities in a project subject to precedence, time, and resource constraints. The basic scheduling algorithm accounts for time constraints and nonstandard precedences, as well as holiday and calendar information. The resource allocation algorithms use several heuristics to determine feasible schedules subject to very general resource availability profiles. All of these algorithms account for holidays and work shifts and schedule the activities around nonworking periods. The activity and resource calendar specifications can accomodate very complicated work patterns.

The main intent of this paper is to show that the use of the CPM procedure need not be limited to activity or project networks. The architecture of the procedure and its relation to the other components of the SAS System enable you to use it successfully to solve problems in several areas that do not fit the typical project management scenario.

Job shop and flow shop scheduling is one area that lends itself very naturally to solutions using the scheduling heuristics embedded in the CPM procedure. This paper illustrates how to compute some of the standard measures of performance of a schedule using the SAS System and how to implement some of the job shop and flow shop scheduling algorithms using PROC CPM. Most algorithms just determine a sequence of the jobs that are to be performed on each machine, without any reference to actual dates or times. From a practical point of view, however, a foreman or a supervisor needs to know the exact times when each job is to be scheduled on each machine. Using the CPM procedure, you can compute the sequence directly in calendar dates, or, given a desired sequence of jobs, you can compute the corresponding calendar dates for a particular time span. Once the schedule is computed, it can be displayed graphically using the GANTT or NETDRAW procedures.

Finally, this paper uses a manpower scheduling example to show some of the new features of the CPM procedure that allow resource-driven durations and resource calendars.

Job Shop and Flow Shop Scheduling

Typical manufacturing systems produce a variety of products, each requiring a specified set of operations on specific machines or workstations, in a predetermined order. If each product requires the same machines in the same sequence, the system is called a *flow shop*. On the other hand, each product (or *job*) in a *job shop* may require a different order of processing on the different machines. The problem is to determine the order in which the various jobs are to be processed on the different machines to optimize the performance of the system. Several standard performance measures for evaluating optimality are defined in the next section. *Scheduling* is the allocation of the machines to the different tasks required for the production of the different products.

Scheduling problems can get very complex, even with a small number of jobs and machines. For example, if *n* jobs are to be processed on a single machine, there are *n*! ways in which they can be ordered (or *sequenced*); this number grows very rapidly. For *single-machine* or *two-machine* problems, the optimal sequence of jobs is known for a variety of performance measures. When there are more than two machines, however, optimal solutions are nearly impossible to determine in a reasonable amount of time, and heuristic methods are used to obtain feasible solutions that have desirable properties. For a detailed analysis of the subject, refer to Baker (1974).

A common heuristic for most scheduling problems is to use some type of a *dispatching* rule. A *dispatching* rule examines the set of operations that are available for execution at any given time and chooses the next operation using some specific priority rule. The resource allocation algorithm in the CPM procedure can be used to implement several heuristic dispatching rules. To use this, you need to reconcile the two different sets of terminology, the one used in scheduling literature and the other used in a project management framework. Most of the quantities used in the

scheduling heuristics are saved in the Schedule data set produced by PROC CPM or can be easily computed from the information stored there.

The next section defines some standard scheduling terms and the corresponding quantities in the project setting. Next, some simple single machine and parallel machine problems are used to illustrate how to determine the job schedule and compute some measures of performance. A more general job shop scheduling example is then followed by a yarn production example that incorporates several production constraints.

Definition of Terms

As a first step in establishing a correspondence between the terminology used in the two disciplines, the basic components of job shop or flow shop scheduling are identified with the corresponding terms in project management. The individual *operation* of a given job on a given machine is identified with an *activity*, the fixed sequence of operations for a given job determines the *precedence constraints* between the activities, and the *machines* are the *resources*. Then the determination of a schedule for the operations of each job on each machine is the same as solving a resource allocation problem in project management using the CPM procedure.

Next, for a general job shop where each job consists of a sequence of operations on different machines, consider the various input parameters, some performance measures, and certain common dispatching rules (Baker 1974). Lowercase letters represent input parameters and uppercase letters are used for quantities that are a function of the scheduling rule used.

Processing time t_{ij}*:* The amount of processing required by operation *i* of job *j*. This is just the *duration* of an activity.

Due date d_j*:* The time at which job *j* is required to be completed. This can be specified using an *alignment date* or a time constraint for the last operation required for the job.

Completion time C_j*:* The time at which job *j* is completed. This is the job's *resource constrained finish time*, saved in the S_FINISH variable in the Schedule output data set produced by PROC CPM. Note that the job's resource constrained finish time is the resource constrained finish time of the *last* operation for the job.

Flowtime F_j*:* The amount of time job *j* spends in the system; this includes the waiting time and the processing time. This is the difference between the job's *resource constrained finish time* (S_FINISH) and its *early start time* (E_START).

Lateness L_j*:* The amount of time by which the completion time of job *j* exceeds its due date, namely, $C_j - d_j$. Note that if an operation is early, its lateness is negative.

Tardiness T_j*:* The amount of time by which a late job *j* is late; for operations that are early or on time, tardiness is defined to be 0. Thus, $T_j = max(0, L_j)$.

The quantities F_j, L_j, and T_j are important measures for evaluating schedules. From the definitions, you can see that these measures are easily computed from the schedule produced by the CPM procedure. Often, these measures are aggregated over all the jobs in the system, and different schedules are compared using a single performance measure, such as *average flowtime*, *mean tardiness*, *maximum flowtime*, *maximum tardiness*, or *number of tardy jobs*. Since all the required times are readily computed from the Schedule output data set, the aggregate measures can also be determined using the SAS System (PROC MEANS or PROC SUMMARY).

One other important measure of overall performance of a schedule is the *makespan*, which is denoted by *M* and equals the length of time required to complete all jobs. Thus, $M = max(C_j)$.

Several scheduling heuristics attempt to minimize at least one of these measures of performance using dispatching rules that depend on the state of the system at the given time. Some standard dispatching rules are:

Shortest Processing Time (**SPT**): Select an operation with the minimum processing time.

Longest Processing Time (**LPT**): Select an operation with the maximum processing time.

Earliest Due Date (**EDD**): Select an operation associated with the job that has the earliest due date.

Most Work Remaining (**MWKR**): Select the operation associated with the job having the most work remaining to be processed.

Random (**RANDOM**): Select an operation at random.

Slack Rule (**SLACK**); Select an operation that has minimum slack time.

The next sections provide examples of implementing some of these rules.

Single-Machine Model

The basic single-machine problem consists of a set of *n* independent single-operation jobs to be scheduled on a single machine. Optimal sequencing rules are known for several of the performance measures, and these sequences can often be determined simply by sorting the jobs according to the required criterion. However, this paper uses PROC CPM to solve this simple example to illustrate how the same method can be used for more general problems.

Table 1. Single-Machine Problem

Job	Processing Time	Due Date
A	3	16
B	4	11
C	5	17
D	6	12

Consider the set of jobs shown in Table 1 with the corresponding processing times and due dates (Salvendy 1982). Since each job has only one operation, the two terms are

used interchangeably for the single-machine case. The problem is to determine the optimal sequence of these jobs.

To use PROC CPM, create an Activity data set and a Resource data set (SAS Institute Inc. 1993b), as shown in Output 1. Note that since there are no precedence constraints in this problem, the SUCC variable is missing for all the activities. The due dates are modeled with an alignment value and an alignment type **`fle`** (*finish less than or equal to*). The single machine is treated as a resource, MACHINE, required by each job. The Resource data set indicates that only one machine is available from the beginning (**`per=0`**).

Output 1. Input Data Sets for Single-Machine Problem

Activity Data Set FLSHOP1

OBS	JOB	SUCC	DUR	ALVALUE	ALTYPE	MACHINE
1	A		3	16	fle	1
2	B		4	11	fle	1
3	C		5	17	fle	1
4	D		6	12	fle	1

Resource Data Set SINGMACH

OBS	PER	MACHINE
1	0	1

Some of the scheduling rules can be implemented using information in the input data set. For example, to use the **SPT** rule, invoke PROC CPM using the SHORTDUR scheduling rule. The SAS statements are shown below; the resulting schedule is contained in the S_START and S_FINISH variables, printed in Output 2. Note that the jobs are scheduled in the order A, B, C, and D; the *Lateness* L_j for the jobs are -13, -4, -5, and 6, respectively. This rule minimizes the *average lateness* but may result in some jobs being very late.

```
proc cpm data=flshop1 resin=singmach
     date=0 out=sptone;
   act job;
   succ succ;
   dur dur;
   res machine / per=per rule=shortdur;
   id alvalue;
run;
```

Output 2. Single Machine Flow Shop: **SPT** rule

JOB	DUR	ALVALUE	S_START	S_FINISH
A	3	16	0	3
B	4	11	3	7
C	5	17	7	12
D	6	12	12	18

To sequence the jobs according to the **EDD** rule, use the scheduling rule ACTPRTY with the alignment variable ALVALUE specified as the activity priority variable. In other words, invoke PROC CPM as above, but replace **`rule=shortdur`** with the two options **`rule=actprty`** and **`actprty=alvalue`**. The resulting schedule (not shown) sequences the jobs in the order B, D, A, and C, with corresponding lateness -7, -2, -3, and 1. The **EDD** rule minimizes the *maximum lateness* but not necessarily the *average lateness*.

Some of the other rules require measures that are not available in the input data set but are measures that depend on the unconstrained schedule of the jobs. The unconstrained schedule refers to the schedule of the jobs that assumes an infinite number of machines; in other words, each job is scheduled at its earliest possible time (in this case, at time 0). For example, the **SLACK** rule requires the computation of the Total Float. First, invoke PROC CPM without resource information, as shown below. The resulting Schedule data set, FLSCHED1, is shown in Output 3.

```
proc cpm data=flshop1 date=0 fbdate=17
     out=flsched1 xfervars;
   act job;
   dur dur;
   succ succ;
   alignmdate alvalue;
   aligntype altype;
   id machine;
run;
```

Output 3. Unconstrained Schedule of Jobs

Schedule Data Set FLSCHED1

OBS	JOB	SUCC	DUR	MACHINE	ALVALUE	ALTYPE	E_START	L_START	E_FINISH	L_FINISH
1	A		3	1	16	fle	0	3	13	16	13	14
2	B		4	1	11	fle	0	4	7	11	7	13
3	C		5	1	17	fle	0	5	12	17	12	12
4	D		6	1	12	fle	0	6	6	12	6	11

Next, invoke PROC CPM using the data set FLSCHED1 as input, with the variable T_FLOAT (Total Float) specified as the activity priority variable. The SAS statements follow; the jobs are scheduled in the order D, B, C, and A (see Output 4).

```
proc cpm data=flsched1 resin=singmach
     date=0 out=slackone;
   act job;
   succ succ;
   dur dur;
   res machine / per=per rule=actprty
                 actprty=t_float;
run;
```

Output 4. Single Machine Flow Shop: **SLACK** rule

JOB	DUR	ALVALUE	S_START	S_FINISH
A	3	16	15	18
B	4	11	6	10
C	5	17	10	15
D	6	12	0	6

Note that, since there are no precedence constraints and no time constraints on the start time of the jobs, **`E_START=0`** for all the jobs, and, hence, **`T_FLOAT=L_START`**. Consequently, the **SLACK** rule could also have been implemented using the *original* Activity data set, FLSHOP1, and the LST scheduling rule (namely, RULE=LST). However, the example illustrates the implementation of the **SLACK** rule in a

more general setting where E_START may not be the same for all activities.

Finally, for the single-machine problem, three different measures of performance (WAITTIME, FLOWTIME, and LATENESS) are computed for the three rules (**SPT**, **EDD**, and **SLACK**) using the respective output data sets (and a simple invocation of PROC SUMMARY). The aggregated measures are shown in Output 5. Note that the **SPT** rule is optimal for all three measures.

Output 5. Average Performance Measures

RULE	WAITTIME	FLOWTIME	LATENESS
SPT	5.50	10.00	-4.00
EDD	6.75	11.25	-2.75
SLACK	7.75	12.25	-1.75

Parallel-Machine Model

The parallel-machine model is an extension of the single-machine model where there are *m* identical machines available for processing. Each job can be scheduled on any one of the available machines. To model this situation with the CPM procedure, you use *alternate resources*. A simple example illustrates a heuristic that attempts to minimize the *makespan* using alternate (or substitutable) resources.

Consider a set of eight jobs with processing times {1, 2, .. , 8}, and *m*=3 identical machines. A simple heuristic uses **LPT** scheduling as a dispatching rule (Baker 1974, Algorithm 5.2). The procedure consists of listing the jobs in **LPT** order; at each possible time, assign the next job in the list to the machine that has had the minimum total processing time thus far.

Output 6. Input Data Sets for Parallel-Machine Problem

Activity Data Set FLSHOP2

OBS	JOB	SUCC	DUR	MACHINE	MACH1	MACH2	MACH3	APRTY
1	1		1	1	.	.	.	-1
2	2		2	1	.	.	.	-2
3	3		3	1	.	.	.	-3
4	4		4	1	.	.	.	-4
5	5		5	1	.	.	.	-5
6	6		6	1	.	.	.	-6
7	7		7	1	.	.	.	-7
8	8		8	1	.	.	.	-8

Resource Data Set PARMACH

OBS	PER	OTYPE	RESID	MACHINE	MACH1	MACH2	MACH3
1	0	reslevel		0	1	1	1
2	.	altrate	machine	.	1	1	1

The input data sets are printed in Output 6. Note that the scheduling rule ACTPRTY in the CPM procedure orders the activities in *increasing* order of the activity priority variable; hence, to obtain the **LPT** ordering of the jobs, create a variable, such as APRTY = (-DUR), in the Activity data set, and the jobs will be scheduled in the right order by PROC CPM. To model the parallel machine scenario, assume that each job requires the single resource MACHINE, which has three possible alternatives, MACH1, MACH2, and MACH3.

In the Resource data set, PARMACH (Output 6), the availability for MACHINE is set to 0, while it is set to 1 for the other three resources. The second observation in the Resource data set indicates that the three resources, MACH1, MACH2, and MACH3, are possible alternatives for the resource MACHINE.

The following statements invoke PROC CPM and save the result in the data set LPTSCHED; the relevant information from this data set is printed in Output 7. The variables UMACH1, UMACH2, and UMACH3 indicate which of the three machines is actually used for each job. For example, JOB 8 is scheduled at time 0 on machine 1.

```
proc cpm data=flshop2 resin=parmach
     date=0 out=lptsched;
   act job;
   succ succ;
   dur dur;
   res machine mach1
       mach2 mach3 / per=per rule=actprty
                     actprty=aprty
                     obstype=otype
                     resid=resid;
run;
```

Output 7. Schedule Produced by **LPT** Heuristic

JOB	UMACHINE	UMACH1	UMACH2	UMACH3	S_START	S_FINISH
1	.	.	1	.	11	12
2	.	1	.	.	11	13
3	.	1	.	.	8	11
4	.	.	1	.	7	11
5	.	.	.	1	6	11
6	.	.	.	1	0	6
7	.	.	1	.	0	7
8	.	1	.	.	0	8

```
/* Identify the Machine used for each job */
data machschd;
   set lptsched;
   keep job resource s_start s_finish;
   if umach1=1 then resource="Machine 1";
   if umach2=1 then resource="Machine 2";
   if umach3=1 then resource="Machine 3";

proc sort;
   by resource s_start;

data machgnt;
   set machschd;
   retain segmt_no; /* segment information */
   if resource = lag(resource)
   then segmt_no=segmt_no+1;
   else segmt_no=1;
   /* set x-coordinate for label */
   xpos=(s_start+s_finish)/2.0;
run;
```

The GANTT procedure can graphically present the schedule of the jobs on each machine. It is customary to show the schedules for all the jobs on a given machine on the same horizontal bar. To draw such a Gantt chart, treat the schedules corresponding to the different jobs on the same machine as consecutive segments of a resource constrained schedule. The simple SAS program, shown above, creates the required data set, MACHGNT, from the Schedule ouput data set, LPTSCHED.

The resulting data set is printed in Output 8, which also contains a Label data set, LABDATA, which is used to annotate the job number on each segment. The LABDATA= option, a new feature in PROC GANTT, is used to annotate text on the Gantt chart (SAS Institute Inc. 1994). The *x* coordinate for the job number is set to XPOS, which is computed as the midpoint of the bar corresponding to the job's schedule.

Output 8. Input Data Sets for PROC GANTT

Data Set MACHGNT

OBS	JOB	S_START	S_FINISH	RESOURCE	SEGMT_NO	XPOS
1	8	0	8	Machine 1	1	4.0
2	3	8	11	Machine 1	2	9.5
3	2	11	13	Machine 1	3	12.0
4	7	0	7	Machine 2	1	3.5
5	4	7	11	Machine 2	2	9.0
6	1	11	12	Machine 2	3	11.5
7	6	0	6	Machine 3	1	3.0
8	5	6	11	Machine 3	2	8.5

Label Data Set LABDATA

OBS	_Y	_XVAR	_LVAR	_YOFFSET	_JLABEL
1	-1	xpos	job	0.8	center

The statements invoking PROC GANTT are shown below and the resulting Gantt chart is shown in Figure 1.

```
pattern1 v=e r=8;
title h=1.5 'Machine Schedule: LPT Rule';
proc gantt graphics
     data=machgnt labdata=labdata;
   id resource;
   chart / nolegend nojobnum skip=2
           increment=2;
run;
```

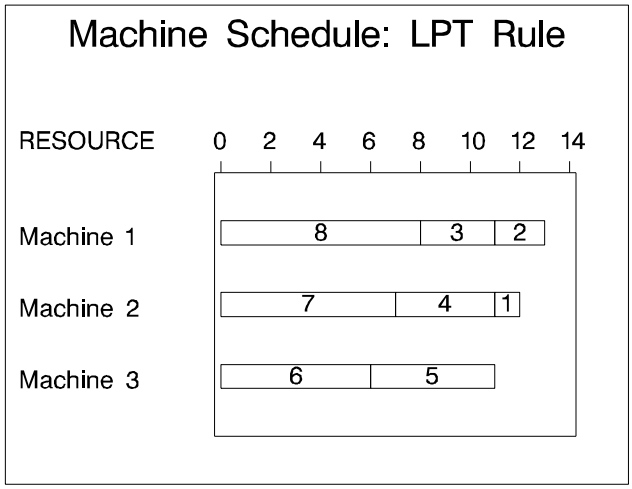

Figure 1. Gantt Chart for Parallel-Machine Model

General Job Shop Model

The previous sections illustrated various scheduling priorities and alternate resources using the CPM procedure. The single-machine and the parallel-machine models did not require the use of any precedence constraints. For more general flow shop and job shop problems, the operations required for each job have a specific order, necessitating the use of precedence constraints.

This section uses a simple job shop example (Baker 1974) to show how to model the various aspects of the problem. Consider a set of four jobs, each requiring one operation on each of three machines. The order in which the machines are required is different for each job. Figure 2 shows the operations for each job, in the required order. Each node represents a single operation and shows the machine required and the processing time on that machine.

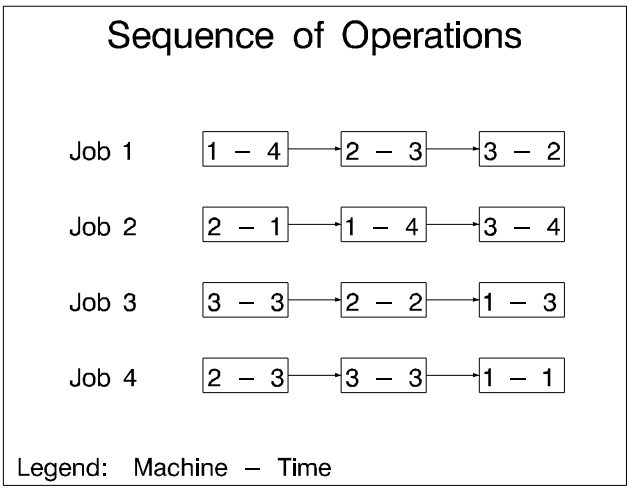

Figure 2. Job Shop Example

Output 9. Input Data for Job Shop Model

Activity Data Set JOBSHOP

OBS	JOB	JOBOP	SUCC	DUR	MACH1	MACH2	MACH3
1	1	111	122	4	1	.	.
2	1	122	133	3	.	1	.
3	1	133	.	2	.	.	1
4	2	212	221	1	.	1	.
5	2	221	233	4	1	.	.
6	2	233	.	4	.	.	1
7	3	313	322	3	.	.	1
8	3	322	331	2	.	1	.
9	3	331	.	3	1	.	.
10	4	412	423	3	.	1	.
11	4	423	431	3	.	.	1
12	4	431	.	1	1	.	.

Resource Data Set THREEMAC

OBS	PER	MACH1	MACH2	MACH3
1	0	1	1	1

The Activity and Resource data sets representing this model in a project management setting are shown in Output 9. The variable JOBOP identifies each operation using a triplet that specifies the job number, operation number, and machine required by that operation. Thus, 412 specifies that the first operation of JOB 4 requires machine 2. The DUR variable specifies the processing time for each operation. The variables MACH1, MACH2, and MACH3 identify the machine required for each operation, and the SUCC variable is used to model the precedence constraints by specifying the immediate successor for each operation.

Heuristics play an important role in schedule generation for the job shop problem because of the highly combinatorial nature of the set of possible schedules. One such heuristic uses the **MWKR** dispatching rule and can be implemented using the unconstrained schedule produced by PROC CPM to generate the required priority variable. Note that the *unconstrained schedule* refers to the *early start* schedule of the operations, which assumes that there are no resource limitations.

The *work remaining* (or **WKR**) for a job is defined as the total duration of all the operations that are still to be processed for that job. Thus, for the three operations comprising JOB 1, the **WKR** at the start of each operation is 9, 5, and 2, respectively. Note that, for each operation, this is simply the difference between *the operation's* early start time (E_START) and the early finish time (E_FINISH) of the *last operation* for the associated job.

The following program uses PROC CPM to find the unconstrained schedule of the operations and then computes the **WKR** for each operation. The resulting data set, WKRJOBS, is printed in Output 10. Once again, the activity priority variable, APRTY, is set to (-WKR) because the dispatching rule uses *decreasing* order of **WKR**.

```
proc cpm data=jobshop out=jobsched;
   act jobop;
   succ succ;
   dur dur;
   id job mach1 mach2 mach3;

/* order operations for each job */
/* from the last to the first    */
proc sort data=jobsched;
   by job descending e_finish;

data wkrjobs;
   set jobsched; retain jobfin;
   /* save finish time of the last */
   /* operation for each job       */
   if job ^= lag(job) then jobfin=e_finish;
   wkr = jobfin - e_start ; aprty = -wkr;
   if mach1=1 then resource="Machine 1";
   if mach2=1 then resource="Machine 2";
   if mach3=1 then resource="Machine 3";
run;
```

Output 10. Activity Data Set for MWKR Rule

Activity Data Set WKRJOBS

JOB	SUCC	DUR	E_START	E_FINISH	LATE_FINISH	APRTY	RESOURCE
1	133	.	2	1	.	-17	9	7	9	7	9	0	0	9	2	-2	Machine 3
2	122	133	3	1	.	1	.	4	7	4	7	0	0	9	5	-5	Machine 2
3	111	122	4	1	1	.	.	0	4	0	4	0	0	9	9	-9	Machine 1
4	233	.	4	2	.	.	1	5	9	5	9	0	0	9	4	-4	Machine 3
5	221	233	4	2	1	.	.	1	5	1	5	0	0	9	8	-8	Machine 1
6	212	221	1	2	.	1	.	0	1	0	1	0	0	9	9	-9	Machine 2
7	331	.	3	3	1	.	.	5	8	6	9	1	1	8	3	-3	Machine 1
8	322	331	2	3	.	1	.	3	5	4	6	1	0	8	5	-5	Machine 2
9	313	322	3	3	.	.	1	0	3	1	4	1	0	8	8	-8	Machine 3
10	431	.	1	4	1	.	.	6	7	8	9	2	2	7	1	-1	Machine 1
11	423	431	3	4	.	.	1	3	6	5	8	2	0	7	4	-4	Machine 3
12	412	423	3	4	.	1	.	0	3	2	5	2	0	7	7	-7	Machine 2

To produce the schedule corresponding to the **MWKR** dispatching rule, invoke PROC CPM again, as shown below. The scheduling rule is specified to be ACTPRTY, with variable APRTY as the activity priority variable; a secondary scheduling rule, RULE2 = SHORTDUR, is specified for breaking ties. The resulting schedule is displayed on the Gantt chart shown in Figure 3.

```
proc cpm data=wkrjobs resin=threemac
     out=mwkrschd;
   act jobop;
   succ succ;
   dur dur;
   res mach1 mach2 mach3 / per=per
                           actprty=aprty
                           rule=actprty
                           rule2=shortdur;
   id job resource;
run;
```

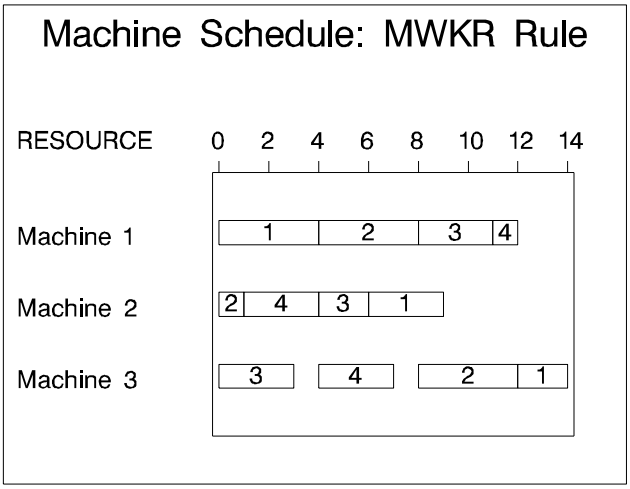

Figure 3. Gantt Chart for Job Shop Model

General Modeling Example

The previous examples illustrated the implementation of several standard scheduling heuristics using the CPM procedure. In all these examples, the data specifying the operations and their processing times and related information were assumed to be readily available in data sets in the required format. In typical manufacturing or production environments, a large part of the problem involves setting up the sequence of operations for the various jobs using data that may be scattered in different tables and using prior knowledge of the manufacturing process. Often, there may be several technological constraints that need to be included in the model. For example, in a dyeing plant, it may be preferable to use the dyer for lighter colors before using it for darker ones.

This section illustrates some of these modeling ideas in a yarn manufacturing example. The problem consists of scheduling the various operations required to manufacture several types of yarns of different colors. All the yarn types need to be dyed and dried. The processing times for these operations depend on the size of the yarn package. After the

drying operation, the processing required depends on the yarn type; warp yarns are shipped directly after processing on a "warper" while the other yarns require a "winding" operation followed by "packing" before "shipping."

Output 11. Yarn Requirements Data

Yarn Requirements Data

OBS	STYLE	TOTAL	PACK	NOPACK	COLOR	WARP	WARPHRS	WIND	WINDHRS	PACKHRS
1	1	2250	750	3	1234	1	5.1	.	.	.
2	2	300	100	3	4657	1	2.0	.	.	.
3	3	400	200	2	9483	1	3.2	.	.	.
4	4	1200	600	2	2941	.	.	1	1.5	2
5	5	400	100	4	3810	.	.	1	2.1	1

Output 11 shows the data for five yarn styles. Each observation specifies the style number, the total requirement, the size of one package for that style, the number of packs, and the color number. The observation also indicates if the particular yarn is a warp yarn (WARP = 1) and the number of hours required on the warper (WARPHRS), or, if it requires winding (WIND = 1), the number of hours required on the winder (WINDHRS) and for packing (PACKHRS) after that.

Note that each style may have more than one batch (depending on the package size), and each batch generates a sequence of operations that need to be performed. Further, the shop floor has two different sizes of dyers: larger batches need one of the bigger machines while smaller batches can use any of the dyeing machines. The drying time required by each batch depends on the size of the package. There are two warpers and two winders. A simple DATA step (see Appendix 1) can be used to create an Activity data set using the data in Output 11. The activity data corresponding to the first batch for the first style of yarn are shown in Output 12. Unique activity names are assigned to each operation using the style number, batch number, and operation number.

Output 12. Partial Activity Data Set

Style 1 : Batch 1

NAME	ACT	SUCC	DUR	COLOR	PACKSIZE	BIG	SMALL	DRYER	WARPER	WINDER
dye	a111	a112	3.00	1234	750	1	.	.	.	.
dry	a112	a113	3.75	1234	750	.	.	1	.	.
warp	a113	a114	5.10	1234	750	.	.	.	1	.
ship	a114		0.00	1234	750	.	.	.	.	.

PROC NETDRAW can be used to display the sequence of operations for each yarn style. Figure 4 shows the activity network corresponding to two of the colors. Such diagrams can be used to display the generic flow of operations for each job.

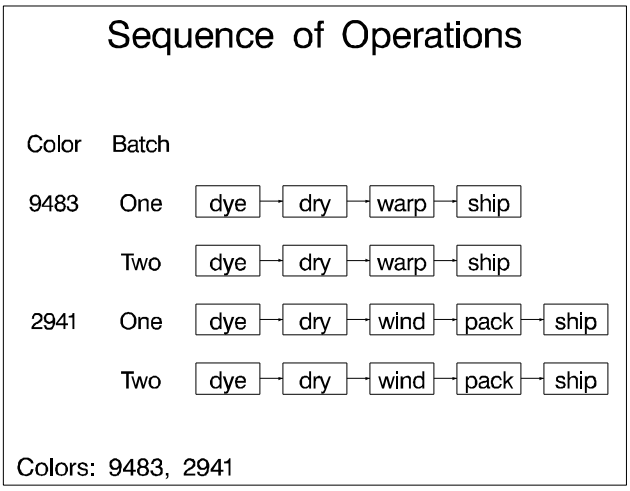

Figure 4. Operation Sequence for Two Styles

The next step is to create a resource availability data set. This data set (printed in Output 13) specifies the availability of the different machines. Note that the resources BIG, SMALL, WINDER, and WARPER are used only to indicate the possible alternate resources, using observations with **TYPE=altrate**. Thus, there are two winders and two warpers available, and the SMALL dyer can be substituted by any of the big or small dyers; however, the observation with **TYPE=ALTPRTY** and **RES=small** indicates that a big dyer should be substituted only if a small one is unavailable.

Output 13. Machine Data

Resource Data Set MACHINES

OBS	TYPE	RES	BIG	SMALL	BIGLG	BIGLL	SMALLA	SMALLL	DRYER	SMALLDE	WINDER	WINDHR	WARPER	WARPHR
1	reslevel	0		0	0	1	1	1	3	0	1	1	0	1	1
2	altrate	.	big	.	.	1	1	.	.	.	.	.	.	.	.
3	altrate	.	small	.	.	1	1	1	1	.	.	.	.	.	.
4	altprty	.	small	.	.	2	2	1	1	.	.	.	.	.	.
5	altrate	.	winder	.	.	.	.	.	.	.	1	1	.	.	.
6	altrate	.	warper	.	.	.	.	.	.	.	.	.	1	1

Calendar Data Set MACHCAL

OBS	_CAL_	_SUN_	_SAT_
1	0	holiday	holiday

Suppose that you wish to obtain the schedule for the shop floor for the week of April 11, 1994, and that the standard work day is from 8 a.m. to 6 p.m. on all five weekdays. The calendar information can be specified using a calendar data set, also printed in Output 13. PROC CPM is invoked with the Activity, Resource, and Calendar data sets, as shown in the following program. The resulting Schedule data set is used to display the schedule (with calendar dates and times) for the dyeing machines in Figure 5.

```
proc cpm data=yrnmodel caledata=machcal
         resin=machines
         daylength="10:00"t
         interval=dthour
         date='11apr94:08:00:00'dt;
   activity act;
   duration dur;
   successor succ;
   resource big big1 big2
            small small1 small2
            dryer winder winder1 winder2
            warper warper1 warper2 /
                   obstype=type
                   period=period
                   resid=res;
   id styleno color packsize nopacks
      name yards batch;
run;
```

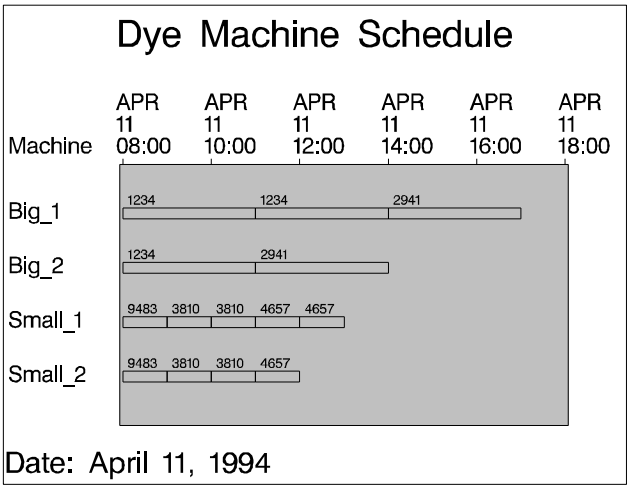

Figure 5. Dyeing Machine Schedule

The preceding yarn example illustrated some of the modeling possibilities using the CPM procedure in conjunction with the capabilities of the SAS language. The CPM procedure provides several other options for resource allocation that can be used effectively in other situations. For instance, several heuristics in the scheduling literature include preemptive dispatching rules; to implement such rules using PROC CPM, you can use the appropriate activity splitting options on the RESOURCE statement. For details regarding the features available in the CPM procedure, refer to SAS Institute Inc. (1993b).

Manpower Scheduling

The previous sections, earlier SUGI papers (Kulkarni 1991 and Kulkarni and Corea 1992), the user's guide (SAS Institute Inc. 1993b), and the Project Management Examples book (SAS Institute Inc. 1993a) illustrated several aspects of the resource scheduling features of the CPM procedure. These included various scheduling priorities, activity splitting, alternate resources, and so forth. All the examples assumed that the duration of an activity is fixed and that all the resources require the same amount of time for the activity. Further, calendars could be associated only with activities.

In projects that use manpower as a resource, it is possible for the same activity to require different amounts of work from different people. Also, the work schedules and vacations may differ for each individual person.

The CPM procedure now supports resource-driven durations and resource calendars. With resource-driven durations, instead of specifying a fixed duration for *all* resources used by an activity, you give the total amount of work required from each resource that the activity uses and the rate of resource usage; the procedure calculates the corresponding duration for each resource used by the activity. Resource calendars allow you to associate calendars with individual resources. Some of these features are illustrated in the following simple example.

Consider a software project requiring two resources: a programmer and a tester. Some of the activities have a fixed duration, requiring the same length of time from both resources; others require a different number of days from the programmer and the tester. Further, some activities require only a fraction of the resource; for example, **`Documentation`** requires only 20 percent of the programmer's time for a total of 2 mandays. The activities in the project, their duration (if fixed) in days, the total work required (if resource-driven) in days, precedence constraints, and the resource requirements are printed in Output 14. A network diagram (Figure 6) displays the activities and their precedence relationships.

Output 14. Project Data

Software Development
Activity Data Set SOFTWARE

ID	ACT	S1	S2	DUR	MANDAYS	PROGRMR	TESTER
Plans & Reqts	1	2	3	2	.	1.0	1.0
Product Design	2	4	5	.	3	1.0	.
Product Design	2	.	.	.	1	.	1.0
Test Plan	3	6	7	3	.	.	1.0
Documentation	4	9	.	.	2	0.2	.
Documentation	4	.	.	.	1	.	0.5
Code	5	8	.	10	.	0.8	.
Test Data	6	8	.	5	.	.	0.5
Test Routines	7	8	.	5	.	.	0.5
Test Product	8	9	.	6	.	0.5	1.0
Finish	9	.	.	0	.	.	.

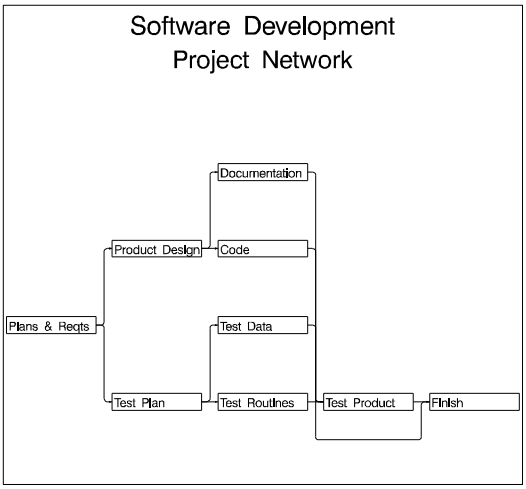

Figure 6. Software Project Network

The following statements invoke PROC CPM with a WORK= specification on the RESOURCE statement, which identifies the amount of work required (in mandays, in this case) from each resource used by an activity. If the WORK variable has a missing value, the activity in that observation is assumed to have a fixed duration. The project is scheduled to start on April 11, 1994, and the activities are assumed to follow a five-day work week. Unlike fixed duration scheduling, each resource used by an activity could have a different schedule; an activity is assumed to be finished only when all of its resources have finished working on it.

```
proc cpm data=software
         out=sftout resout=rout
         resched=rsftout
         date='11apr94'd interval=weekday;
   act act;
   succ s1 s2;
   dur dur;
   res progrmr tester / work=mandays;
   id id;
run;
```

Output 15. Resource Schedule Data

Software Development
Early Schedule for each Resource

ID	ACT	RESOURCE	DUR_TYPE	_WORK_	_DUR_	R_RATE	E_START	E_FINISH
Plans & Reqts	1			2	.	.	11APR94	12APR94
Plans & Reqts	1	PROGRMR	FIXED	2	.	1.0	11APR94	12APR94
Plans & Reqts	1	TESTER	FIXED	2	.	1.0	11APR94	12APR94
Product Design	2			3	.	.	13APR94	15APR94
Product Design	2	PROGRMR	RDRIVEN	3	3	1.0	13APR94	15APR94
Product Design	2	TESTER	RDRIVEN	1	1	1.0	13APR94	13APR94
Test Plan	3			3	.	.	13APR94	15APR94
Test Plan	3	TESTER	FIXED	3	.	1.0	13APR94	15APR94
Documentation	4			10	.	.	18APR94	29APR94
Documentation	4	PROGRMR	RDRIVEN	10	2	0.2	18APR94	29APR94
Documentation	4	TESTER	RDRIVEN	2	1	0.5	18APR94	19APR94
Code	5			10	.	.	18APR94	29APR94
Code	5	PROGRMR	FIXED	10	.	0.8	18APR94	29APR94
Test Data	6			5	.	.	18APR94	22APR94
Test Data	6	TESTER	FIXED	5	.	0.5	18APR94	22APR94
Test Routines	7			5	.	.	18APR94	22APR94
Test Routines	7	TESTER	FIXED	5	.	0.5	18APR94	22APR94
Test Product	8			6	.	.	02MAY94	09MAY94
Test Product	8	PROGRMR	FIXED	6	.	0.5	02MAY94	09MAY94
Test Product	8	TESTER	FIXED	6	.	1.0	02MAY94	09MAY94
Finish	9			0	.	.	10MAY94	10MAY94

The individual resource schedules, as well as each activity's combined schedule, are saved in a Resource Schedule data set, RSFTOUT, requested by the RESSCHED= option on the CPM statement. This output data set is very similar to the Schedule data set and contains the activity variable and all the relevant schedule variables (E_START, E_FINISH, L_START, and so forth). Some of these variables that are of interest are printed in Output 15 (along with the ID variable). For each activity in the project, this data set contains the schedule for the entire activity as well as the schedule for each resource used by the activity. The variable RESOURCE identifies the name of the resource that the observation refers to; the value of the RESOURCE variable is missing for observations that refer to the entire activity's schedule. The variable DUR_TYPE indicates if the resource is a driving resource or if it is of the fixed type.

The variable _DUR_ indicates the duration of the activity for the resource identified in that observation. For resources that are of the "driving" type, the variable _WORK_ shows the total amount of work (in units of the INTERVAL parameter) required by the resource for the activity in that observation. The variable R_RATE shows the rate of usage of the resource for the relevant activity. Note that for driving resources, the variable _DUR_ is computed as (WORK / R_RATE).

These schedules are plotted on a Gantt chart in Figure 7; note that, for the activity **`Product Design`** (**ACT=2**), the resource PROGRMR is a critical resource while the resource TESTER is not. This data set can also be used to print individual Gantt charts for the different resources.

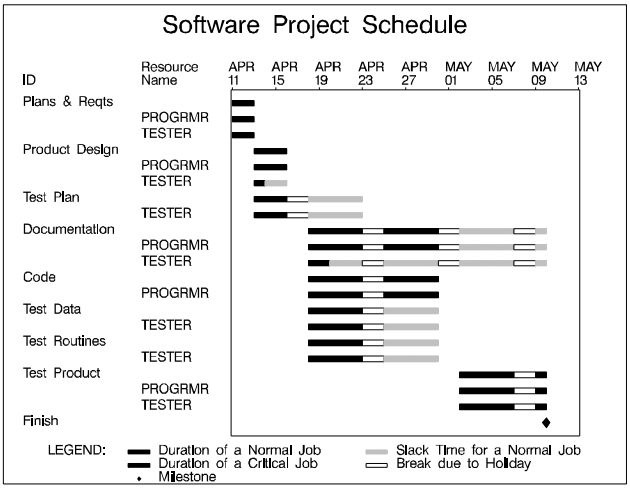

Figure 7. Early and Late Schedules

The daily utilization of the resources is also saved in a data set, ROUT, printed in Output 16. The resource usage data set indicates that you need more than one tester on some days, with both the early schedule (on the 13th, 18th and 19th of April) and the late schedule (on the 6th and 9th of May).

Output 16. Resource Usage Data

Software Development
Resource Usage Data Set ROUT

OBS	_TIME_	EPROGRMR	LPROGRMR	ETESTER	LTESTER
1	11APR94	1.0	1.0	1.0	1.0
2	12APR94	1.0	1.0	1.0	1.0
3	13APR94	1.0	1.0	2.0	0.0
4	14APR94	1.0	1.0	1.0	0.0
5	15APR94	1.0	1.0	1.0	1.0
6	18APR94	1.0	0.8	1.5	0.0
7	19APR94	1.0	0.8	1.5	0.0
8	20APR94	1.0	0.8	1.0	1.0
9	21APR94	1.0	0.8	1.0	1.0
10	22APR94	1.0	0.8	1.0	1.0
11	25APR94	1.0	0.8	0.0	1.0
12	26APR94	1.0	1.0	0.0	1.0
13	27APR94	1.0	1.0	0.0	1.0
14	28APR94	1.0	1.0	0.0	1.0
15	29APR94	1.0	1.0	0.0	1.0
16	02MAY94	0.5	0.7	1.0	1.0
17	03MAY94	0.5	0.7	1.0	1.0
18	04MAY94	0.5	0.7	1.0	1.0
19	05MAY94	0.5	0.7	1.0	1.0
20	06MAY94	0.5	0.7	1.0	1.5
21	09MAY94	0.5	0.7	1.0	1.5
22	10MAY94	0.0	0.0	0.0	0.0

Suppose now that you have only one tester and one programmer. You can determine a resource-constrained schedule using PROC CPM (as in the fixed duration case) by specifying a resource availability data set, RESIN (Output 17).

Output 17. Resource Availability Data

Software Development
Resource Availability Data Set

OBS	PER	OTYPE	PROGRMR	TESTER
1	11APR94	reslevel	1	1

The following statements invoke PROC CPM and the resulting resource constrained schedule is printed in Output 18; the early start schedule is also printed for comparison. Note that the project still finishes on May 10, but some of the activities (3, 4, 6, and 7) are delayed. The resource constrained schedule is plotted on a Gantt chart in Figure 8; both resources follow the same weekday calendar.

```
proc cpm data=software resin=resin
         out=sftout1 resout=rout1
         rsched=rsftout1
         date='11apr94'd interval=weekday;
   act act;
   succ s1 s2;
   dur dur;
   res progrmr tester / work=mandays
                        obstype=otype
                        period=per;
   id id;
run;
```

Output 18. Resource Constrained Schedule

Software Development
Resource Constrained Schedule

ID	A C T	Resource Name	RESOURCE TYPE	DURATION	ES	EF	LS	LF
Plans & Reqts	1				11APR94	12APR94	11APR94	12APR94
Plans & Reqts	1	PROGRMR	FIXED		11APR94	12APR94	11APR94	12APR94
Plans & Reqts	1	TESTER	FIXED		11APR94	12APR94	11APR94	12APR94
Product Design	2				13APR94	15APR94	13APR94	15APR94
Product Design	2	PROGRMR	RDRIVEN		13APR94	15APR94	13APR94	15APR94
Product Design	2	TESTER	RDRIVEN		13APR94	13APR94	13APR94	13APR94
Test Plan	3				14APR94	18APR94	13APR94	15APR94
Test Plan	3	TESTER	FIXED		14APR94	18APR94	13APR94	15APR94
Documentation	4				26APR94	09MAY94	18APR94	29APR94
Documentation	4	PROGRMR	RDRIVEN		26APR94	09MAY94	18APR94	29APR94
Documentation	4	TESTER	RDRIVEN		26APR94	27APR94	18APR94	19APR94
Code	5				18APR94	29APR94	18APR94	29APR94
Code	5	PROGRMR	FIXED		18APR94	29APR94	18APR94	29APR94
Test Data	6				19APR94	25APR94	18APR94	22APR94
Test Data	6	TESTER	FIXED		19APR94	25APR94	18APR94	22APR94
Test Routines	7				19APR94	25APR94	18APR94	22APR94
Test Routines	7	TESTER	FIXED		19APR94	25APR94	18APR94	22APR94
Test Product	8				02MAY94	09MAY94	02MAY94	09MAY94
Test Product	8	PROGRMR	FIXED		02MAY94	09MAY94	02MAY94	09MAY94
Test Product	8	TESTER	FIXED		02MAY94	09MAY94	02MAY94	09MAY94
Finish	9				10MAY94	10MAY94	10MAY94	10MAY94

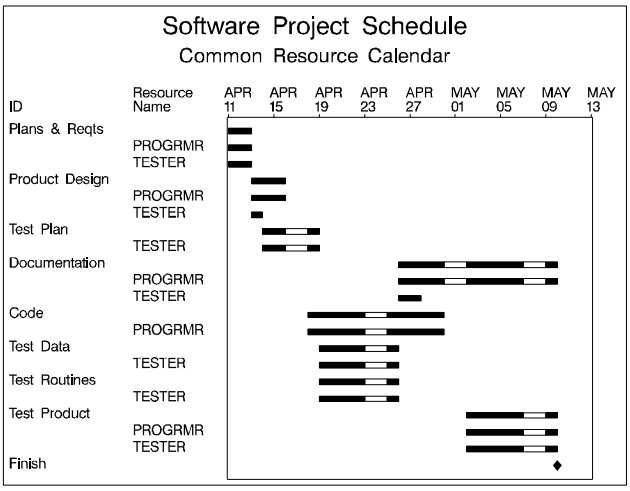

Figure 8. Resource Constrained Schedule

Now, suppose that the tester switches to part-time employment, working only four days a week. Thus, the two resources have different calendars. What effect will this change have on the project schedule? To answer this question, define a calendar data set identifying calendar **1** as having a holiday on Friday (see Output 19). In a new resource availability data set (also printed in Output 19), associate calendar **1** with the TESTER and calendar **0** with the resource, PROGRMR. Note that **0** refers to the default calendar, which is the weekday calendar for this project (since INTERVAL = WEEKDAY).

Output 19. Resource and Calendar Data

Software Project
Calendar Data Set CALENDAR

OBS	_CAL_	_FRI_
1	1	holiday

Resource Data Set RESIN2

OBS	PER	OTYPE	PROGRMR	TESTER
1	11APR94	reslevel	1	1
2	.	calendar	0	1

Next, invoke PROC CPM, as shown below, with the Activity, Resource, and Calendar data sets, to obtain the revised schedule, plotted in Figure 9. Note that the project is delayed by two days because of the TESTER's shorter work week which is illustrated by the longer holiday breaks in the TESTER's schedule bars.

```
proc cpm data=software resin=resin2
         caledata=calendar
         out=sftout2 rsched=rsftout2
         resout=rout2
         date='11apr94'd interval=weekday;
   act act;
   succ s1 s2;
   dur dur;
   res progrmr tester / work=mandays
                        obstype=otype
                        period=per;
   id id;
run;
```

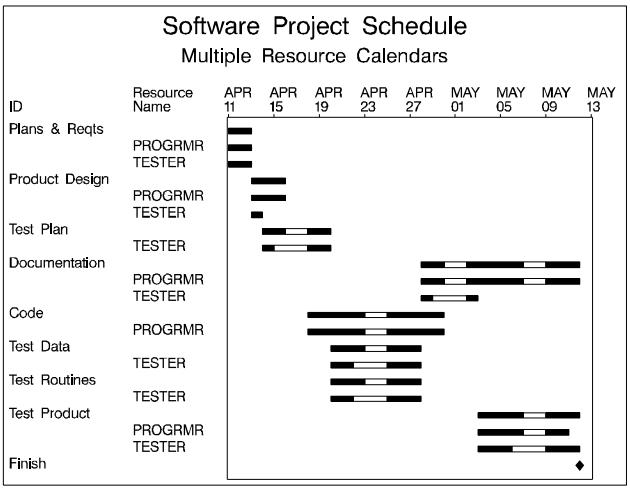

Figure 9. Resource Constrained Schedule

The preceding example illustrated some of the possible scenarios that can be modeled using resource-driven durations and resource calendars. For further details, refer to SAS Institute Inc. (1994).

Conclusion

Though the CPM procedure is primarily designed for determining project schedules subject to precedence, time, and resource constraints, its architecture makes it a flexible and powerful tool for solving problems that arise in a variety of other settings. The output data sets produced by the CPM procedure contain a wealth of information; the trick is to use the data effectively to derive the measures required in other applications. This paper illustrates several such techniques for implementing standard scheduling heuristics for job shop and flow shop problems.

The yarn manufacturing example illustrates the broader modeling ability of the SAS language. Data are usually available in different sources and in formats that are specific to the application. It is easy to convert the available data into the required project data sets and invoke PROC CPM to determine the schedule. A significant advantage of using the CPM procedure in a manufacturing environment is the ability to produce schedules using calendar dates and to account for holidays and complicated work patterns.

Finally, resource driven durations and resource calendars are illustrated using a simple software project. These two features enhance your ability to schedule people more effectively.

References

Baker, K.R. (1974), *Introduction to Sequencing and Scheduling*, New York: John Wiley & Sons, Inc.

Kulkarni, R. (1991), "Scheduling with the CPM Procedure," *Proceedings of the Sixteenth Annual SAS Users Group International Conference*.

Kulkarni, R. and Corea, G. (1992), "Using the Project Management Tools in the SAS/OR® System," *Proceedings of the Seventeenth Annual SAS Users Group International Conference*.

Salvendy, G. (1982), *Handbook of Industrial Engineering*, New York: John Wiley & Sons, Inc.

SAS Institute Inc. (1993a), *SAS/OR® Software: Project Management Examples, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993b), *SAS/OR® User's Guide: Project Management, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1994), *Update to Version 6 SAS/OR User's Guide: Project Management*, Cary, NC: SAS Institute Inc.

Appendix 1

```
/* Make an activity data set from */
/* the yarn requirements table    */

data yrnmodel;
   length act  $ 8;
   length succ $ 8;
   length name $ 8;

   keep act succ name dur styleno batch
        big small dryer warper winder
        color yards packsize nopacks
        big1 big2 small1 small2
        winder1 winder2 warper1 warper2;

   set yarnrqts;

   big1=.; big2=.;
   small1=.; small2=.;
   winder1=.; winder2=.;
   warper1=.; warper2=.;
   warper = .; winder = .;

   /* Each style generates 'nopacks' jobs */
   do i=1 to nopacks;

      batch=i;

      /* Each job visits a dyer and the dryer */
      act='a'||left(put(styleno,1.))
              ||left(put(i,1.))||'1';
      succ= 'a'||left(put(styleno,1.))
              ||left(put(i,1.))||'2';

      /* set the dyer requirement (small or big) */
      /* and duration depending on batch size    */
      name='dye';
      if 500 < packsize le 1000 then do;
         big=1; dur=3;
         end;
      if packsize le 500 then do;
         small=1; dur=1;
         end;
      output;
```

```
big=.; small=.;

act='a'||left(put(styleno,1.))
       ||left(put(i,1.))||'2';
succ= 'a'||left(put(styleno,1.))
         ||left(put(i,1.))||'3';
name='dry';
dryer=1;
dur = packsize/200;
output;
dryer=.;

/* Warped jobs go directly to shipping */
if warp ne . then do;
   act='a'||left(put(styleno,1.))
          ||left(put(i,1.))||'3';
   succ= 'a'||left(put(styleno,1.))
            ||left(put(i,1.))||'4';
   name='warp';
   dur=warphrs; /* warping time */
   warper=1;
   output;
   warper=.;
   act='a'||left(put(styleno,1.))
          ||left(put(i,1.))||'4';
   succ=' ';
   name='ship';
   dur=0;
   output;
   end;

/* Wound jobs get packed before shipping */
else do;
   act='a'||left(put(styleno,1.))
          ||left(put(i,1.))||'3';
   succ= 'a'||left(put(styleno,1.))
            ||left(put(i,1.))||'4';
   name='wind';
   dur=windhrs;
   winder=1;
   output;
   winder=.;
   act='a'||left(put(styleno,1.))
          ||left(put(i,1.))||'4';
   succ= 'a'||left(put(styleno,1.))
            ||left(put(i,1.))||'5';
   name='pack';
   dur=packhrs;
   output;
   act='a'||left(put(styleno,1.))
          ||left(put(i,1.))||'5';
   succ=' ';
   name='ship';
   dur=0;
   output;
   end;
end;
run;
```