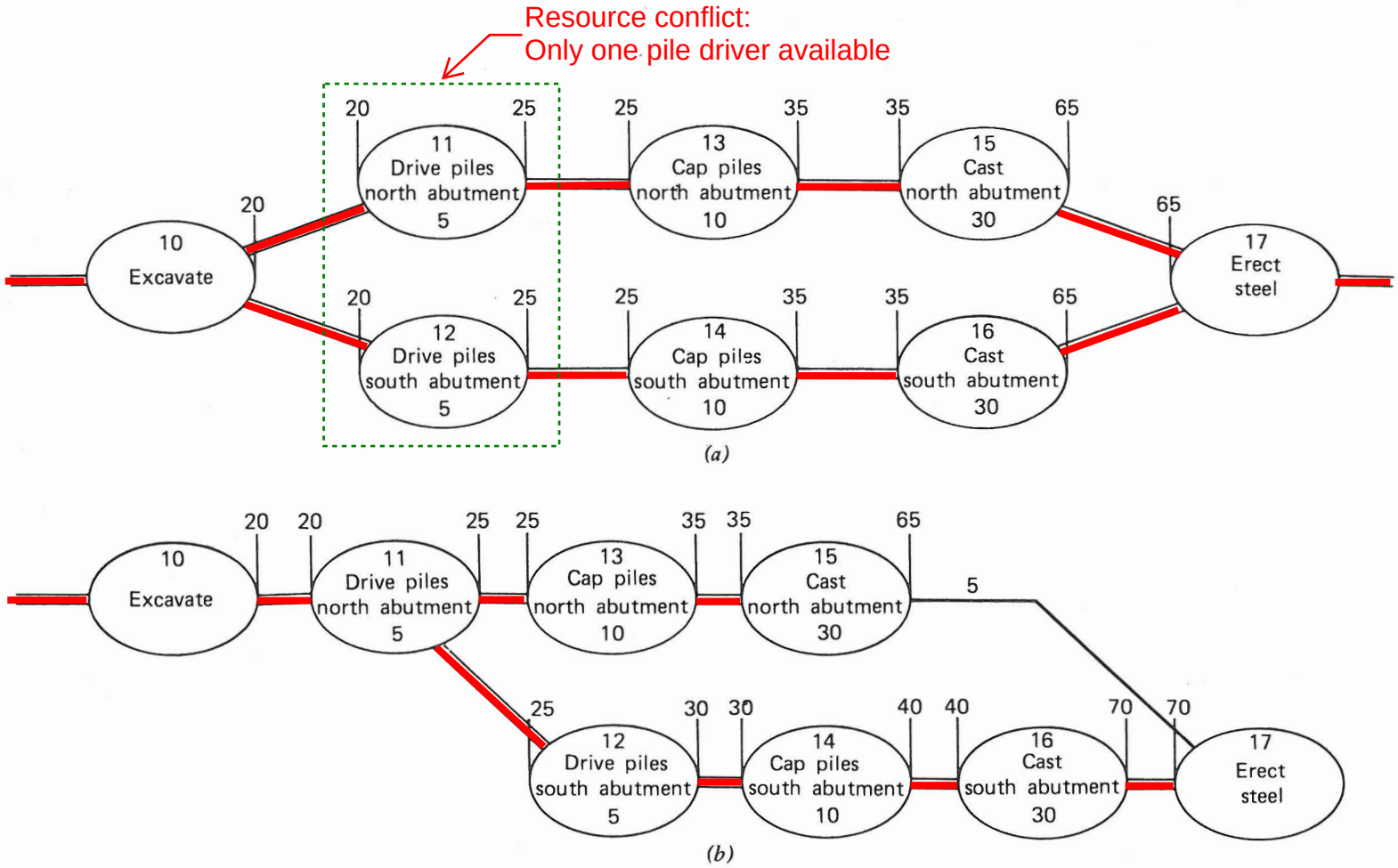


Figure 11.1 Resolving resource conflicts on parallel paths.

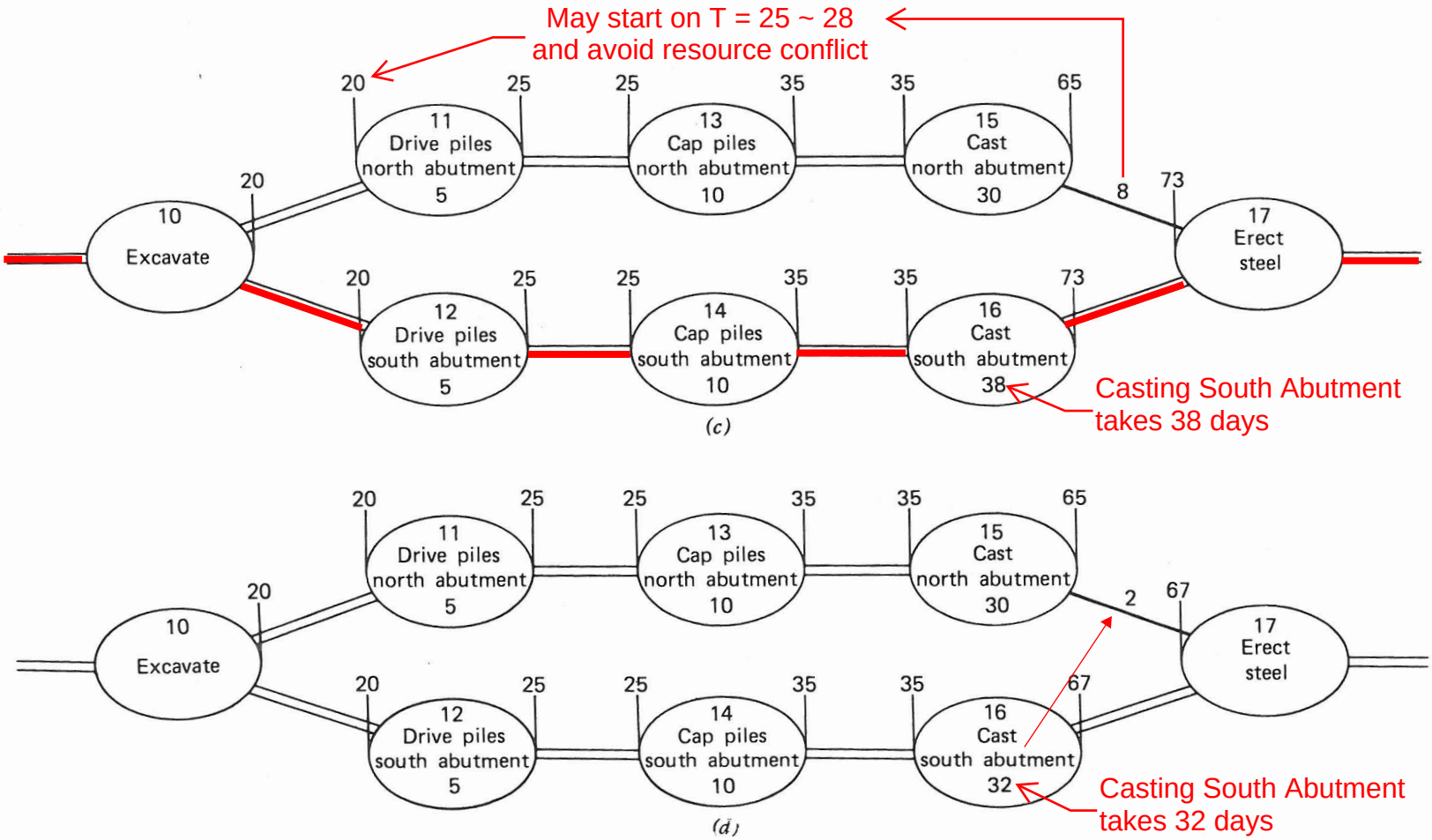
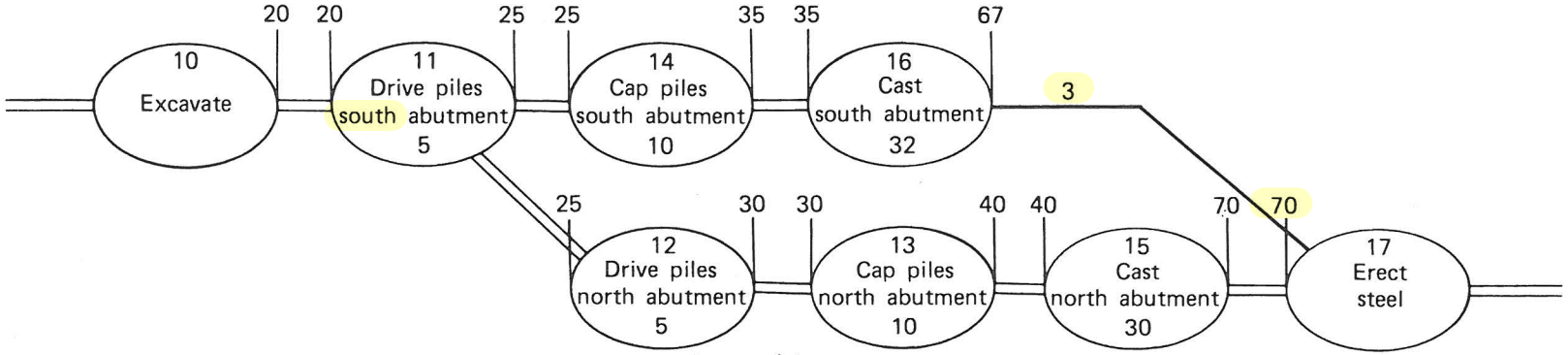


Figure 11.1 (Continued)

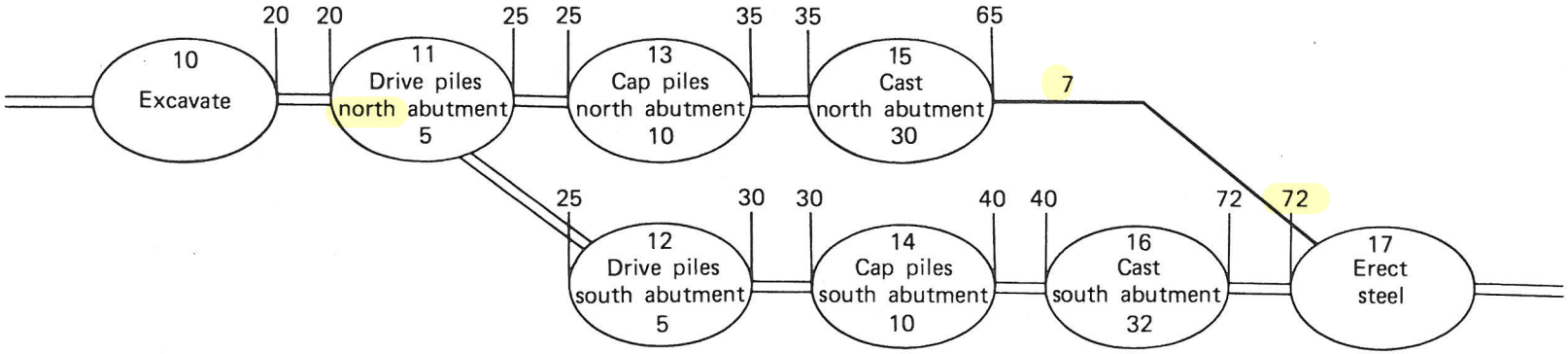
Scheduling Activities Without Calculations
- Parallel activity chains
- With common starts and ends
- Chains are identical (act's & durations)
- One activity in each chain uses identical resource

Efficient. Ends at T=70.



(e)

Inefficient. Ends at T=72.



(f)

Figure 11.1 (Continued)

If A and B
- require the same resource, and
- their early scheduling times overlap
then, there is a resource conflict.
Harder to spot and rectify.
Requires analysis: Limited Resource Allocation.

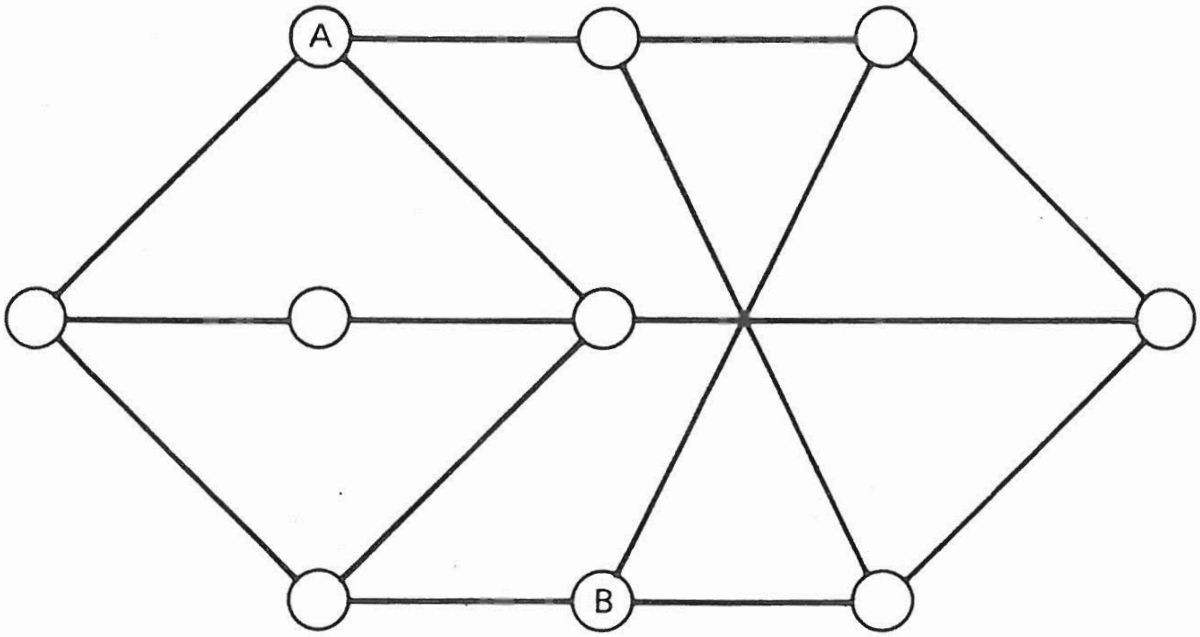


Figure 11.2 Possible resource conflicts.

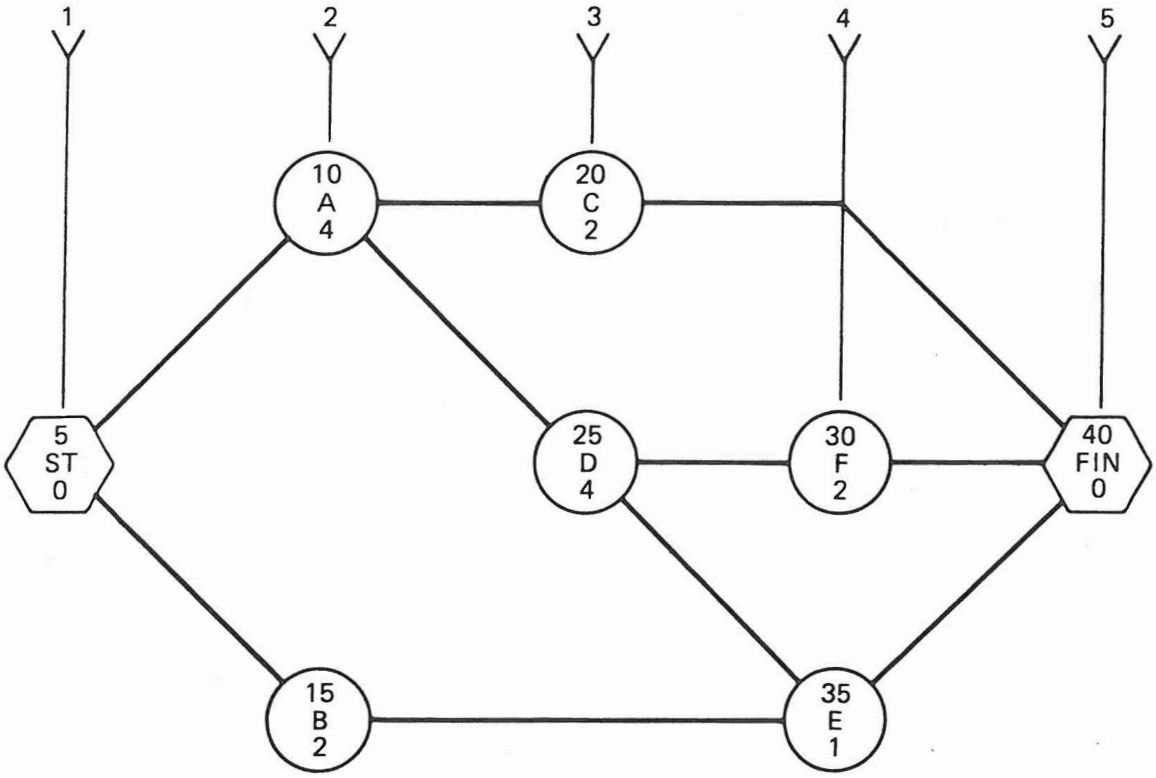


Figure 11.4 Precedence diagram for leveling example.

TABLE 11.1

Early Start Schedule Information

Activity	*T* (1) days	*ESD*	*EFD*	*TF*	*FF*	Res Rate (2) men/day	Res Days (3)=(1)x(2) mandays
A*	4	0	4	0		2	8
B	2	0	2	7	6	2	4
C	2	4	6	4	4	4	8
D*	4	4	8	0		6	24
E	1	8	9	1	1	1	1
F*	2	8	10	0		2	4
						Total	49

Mandays for the entire project (constant) →

*Critical.

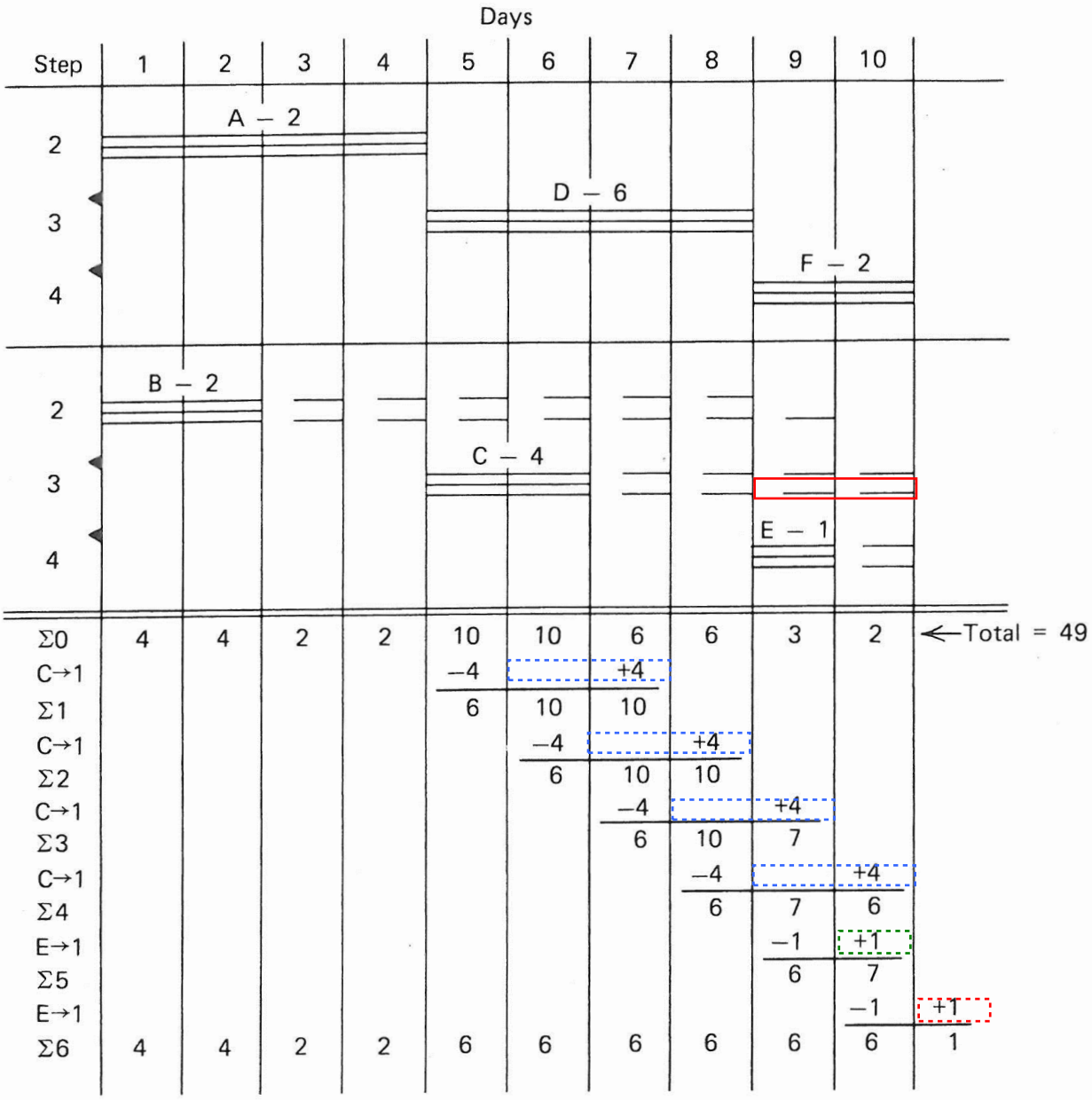


Figure 11.5 Leveling example—traditional approach.

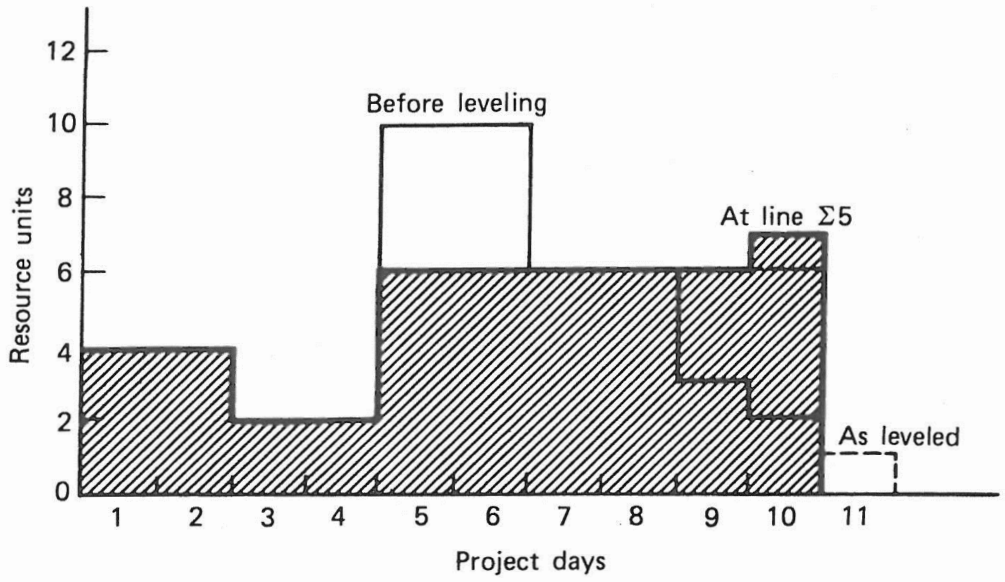


Figure 11.6 Resource histogram—leveling example—traditional approach.

A simple heuristic program

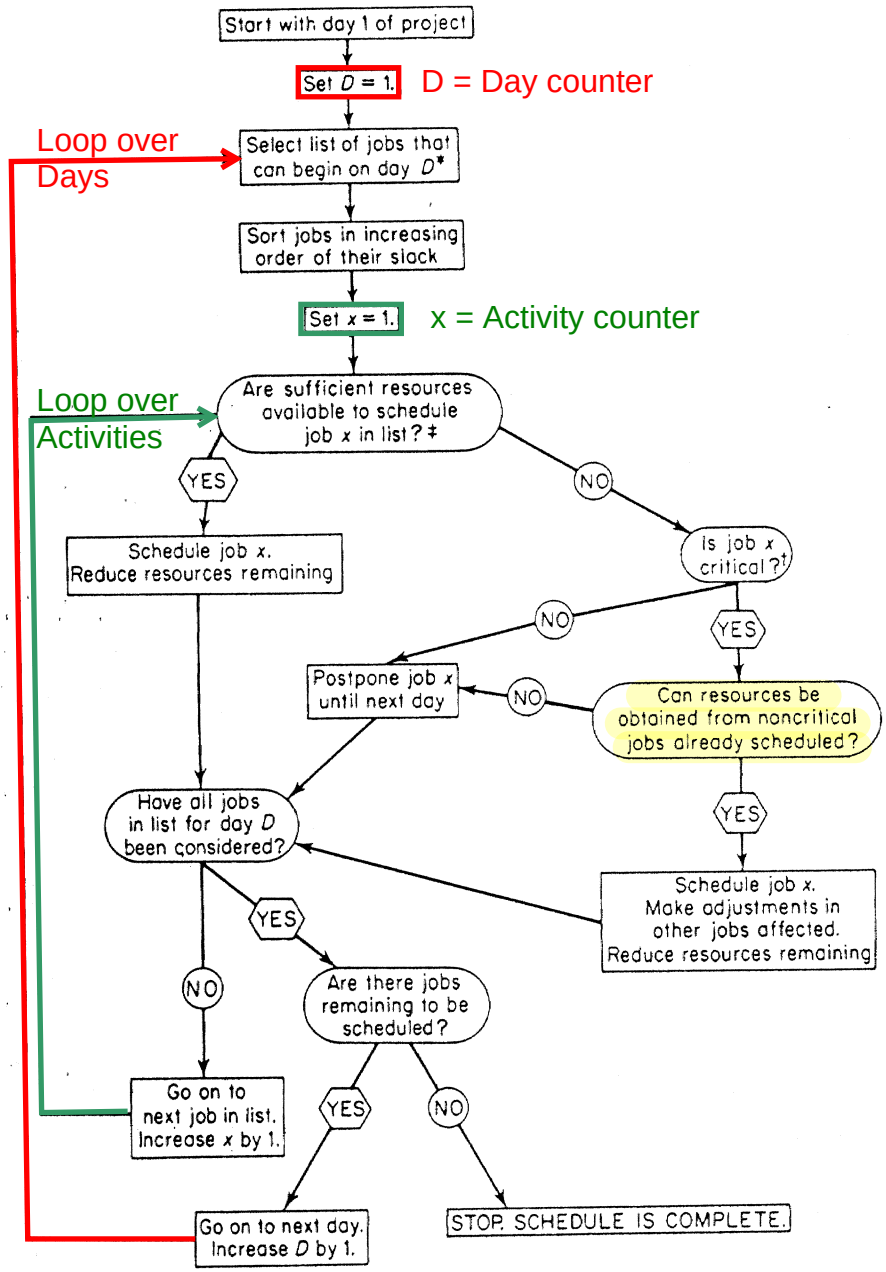
As an illustration of the resource allocation approach to project scheduling, consider the simple program shown in flow-diagram form in Figure 7-8.[4] The program is based essentially on three rules (heuristics):

1. Allocate resources serially in time. That is, start on the first day and schedule all jobs possible, then do the same for the second day, and so on.
2. When several jobs compete for the same resources, give preference to the jobs with the least slack.
3. Reschedule noncritical jobs, if possible, to free resources for scheduling critical or nonslack jobs.

To illustrate how these heuristics operate in the program, let us apply them to a small project consisting of 10 jobs, each of which requires a certain amount of time and a given number of men (crew). For simplicity, let us assume that all the men are interchangeable. In practical situations, however, the various jobs may require men of different skills as well as other resources such as machines, materials, money, and so forth.

Figure 7-9 shows the schedule graph for this small project. All jobs have been started as early as their predecessors will allow. The number above each arrow identifies the job and its crew size. Job 4, for example, requires 4 men on each of days 2, 3, and 4. Thus we can calculate daily manpower requirements for the schedule by summarizing vertically the crew sizes of all jobs active on a given day. In the schedule graph in Figure 7-9, manpower requirements start at 7 on day 1, climb to 13 on days 2 and 3, and so on.

[4]The illustration in this section is adapted from J. D. Wiest's "Heuristic Programs for Decision Making," *Harvard Business Review* (Sept.-Oct. 1966). Used by permission of Harvard Business Review.



*Day D is the day under consideration.
† A job is critical if it has no slack.
‡Job x is the job under consideration.

FIGURE 7-8.

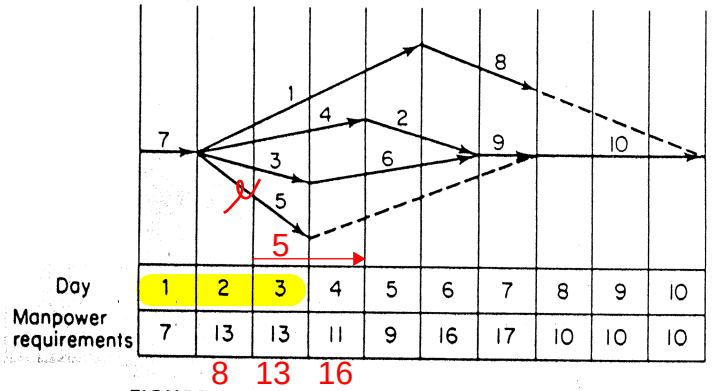


FIGURE 7-9. Schedule Graph, Unlimited Resources

Suppose, however, that only 10 men are available for the project on any one day. How should the project be scheduled so as not to exceed this constraint but still to complete the project as soon as possible? One way to find the shortest feasible schedule would be to enumerate all possible schedules. For example, on day 1 there is just one possible choice of jobs (job 7). On day 2, with four jobs available, there are 13 feasible combinations: four consisting of a single job, six consisting of pairs of jobs, and three consisting of triples of jobs. (The remaining combinations exceed the manpower limit of 10.) Each of these 13 choices represents a different way to begin a schedule, and multiple choices fan out from each of these on succeeding days. The number of all possible schedules is very large, even for this small project.

Day-to-day decisions

The heuristic program diagrammed in Figure 7-8 does not examine all possible schedules; it selects just one combination of jobs for each day, only occasionally retracing its steps to see if a better set could be found. Scheduling day by day (as required by the first heuristic of the program), the program would make the following decisions:

Day 1

Only one job (7) is available to start in this period and there are sufficient men to schedule it.

Schedule job 7 (slack = 0, calculated relative to jobs on the critical path, which are defined as having no slack); 3 men remain available.

Day 2

Four jobs (1, 3, 4, and 5) can be started on day 2, but there are not enough men to schedule them all. The second heuristic calls for scheduling the jobs with the least slack first.

Schedule job 3 (slack = 0); 7 men remain.
Schedule job 4 (slack = 0); 3 men remain.
Schedule job 1 (slack = 3); 2 men remain.

Job 5 must be delayed, as there are just 2 men unassigned.

Postpone job 5 (slack = 4).

Day 3

We assume that jobs cannot be interrupted once started; partially completed jobs have first call on the available resources.

Continue job 3 (slack = 0); 7 men remain.
Continue job 4 (slack = 0); 3 men remain.
Continue job 1 (slack = 3); 2 men remain.

Since there are still not enough men for job 5, it must be delayed again.

Postpone job 5 (slack = 3).

At this point the schedule graph appears as shown in Figure 7-10. Heavy lines indicate jobs already scheduled; light lines are projected schedules, which still must be checked for feasibility. Note that job 5, which has been postponed 2 days, now has only 2 days slack, or possible slippage, remaining.

Day 4: Delay Act-1 to reclaim resources and start Act-6

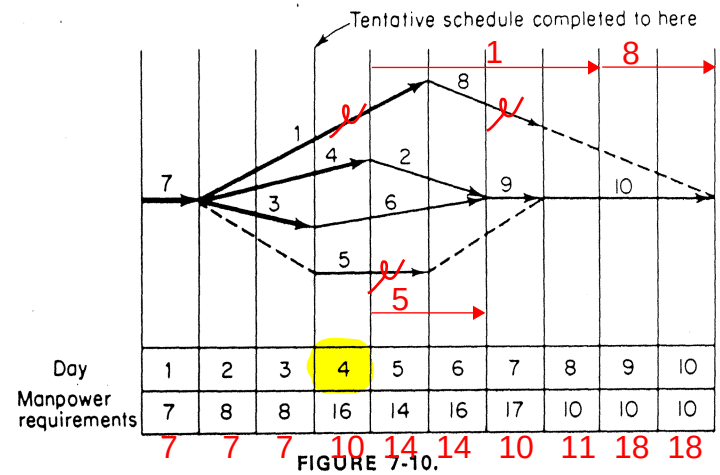


FIGURE 7-10.

Day 4

Continue job 4 (slack = 0); 6 men remain.
Continue job 1 (slack = 3); 5 men remain.
Job 6 has no slack and hence is critical, but only 5 men are still unassigned.

The third heuristic is now brought into play: Are there noncritical jobs still active which could be postponed without delaying the project? Job 1 satisfies this requirement and hence is reassigned to begin later.

Reschedule job 1; 6 men remain.
Schedule job 6 (slack = 0); 0 men remain.
Postpone job 5 (slack = 2).

The schedule graph at the end of day 4 is shown in Figure 7-11.

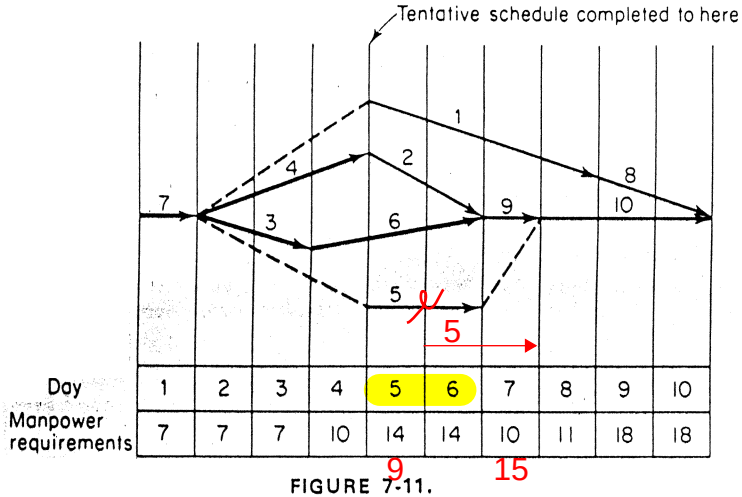


FIGURE 7-11.

Day 5

Continue job 6 (slack = 0); 4 men remain.
Schedule job 2 (slack = 0); 2 men remain.
Schedule job 1 (slack = 0); 1 man remains.
Postpone job 5 (slack = 1).

Day 6

Continue job 6 (slack = 0); 4 men remain.
Continue job 2 (slack = 0); 2 men remain.
Continue job 1 (slack = 0); 1 men remains.
Postpone job 5 (slack = 0).

Job 5 is now critical, but there are no active jobs that can be postponed without delaying the project. Note that the projected finish date is delayed a day by job 5's postponement, as shown in Figure 7-12.

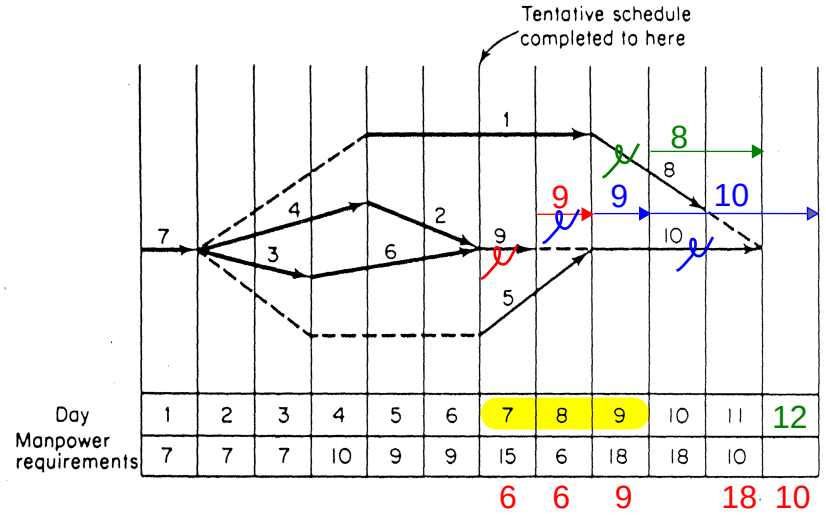


FIGURE 7-12.

Day 7

Continue job 1 (slack = 1); 9 men remain.
Schedule job 5 (slack = 0); 4 men remain.
Postpone job 9 (slack = 1).

Day 8

Continue job 5 (slack = 0); 5 men remain.
Postpone job 9 (slack = 0); no noncritical jobs can be rescheduled.
~~Postpone job 8 (slack = 2).~~

At the end of day 8, the schedule graph appears as in Figure 7-13.

Remaining period

Program decisions for the remaining 6 days are as follows:

Day 9—Schedule job 9 (slack = 0); 1 man remains. Postpone job 8 (slack = 2).
Day 10—Schedule job 10 (slack = 0); no men remain. Postpone job 8 (slack = 1).
Day 11—Continue job 10 (slack = 0); no men remain. Postpone job 8 (slack = 0).
Day 12—Continue job 10 (slack = 1); no men remain. Postpone job 8 (slack = 0).

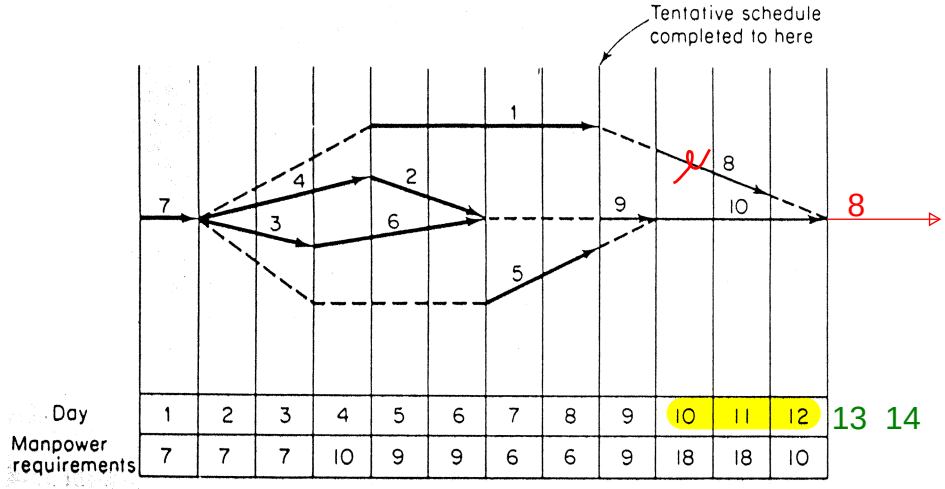


FIGURE 7-13.

Day 13—Schedule job 8 (slack = 0); 2 men remain.
Day 14—Continue job 8 (slack = 0); 2 men remain.

All jobs are now scheduled. The final schedule graph appears in Figure 7-14. Thus the manpower limit of 10 men per day has resulted in a 4-day increase in the project length, compared with the unlimited resources schedule. In this application the heuristic program has found an optimal schedule. There are other feasible schedules, but none is shorter than 14 days.

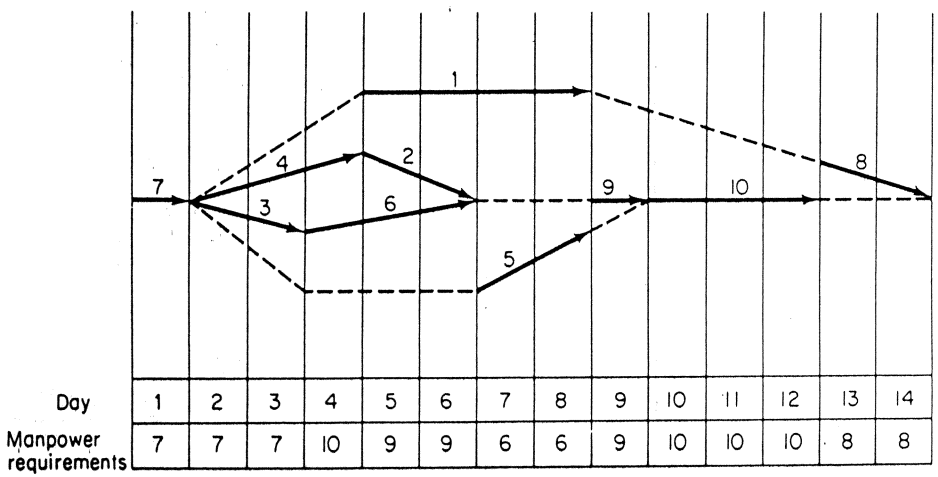


FIGURE 7-14. Final Schedule-Limited Resources

It would not be difficult to think up sample projects for which the above program would *not* produce optimum results. Of course, no heuristic program can *guarantee* an optimum schedule, but the above program could be considerably improved by the addition of appropriate heuristics which take into account more information or special circumstances. For instance, it may be possible in some projects to crash or stretch out certain jobs by varying the rate of resource application. Also, some jobs might allow *splitting*—that is, an interruption in their schedule—while others may have to move to completion once started. Resource limits themselves may vary from period to period. An operational scheduling program should take cognizance of these and other factors in order to be generally useful for project scheduling. Project scheduling situations often differ so markedly, however, that special programs must be written or more general programs modified to meet particular needs. Heuristic programming, as a general technique, facilitates such custom tailoring of scheduling procedures.

- Heuristic, no guarantee of optimality
- Possible improvements:
  - Change the resource rate (i.e, men/day)
    i.e., lengthen or shorten activities
  - Allow splitting (discontinuous activities)
  - Allow variable resource limits over time