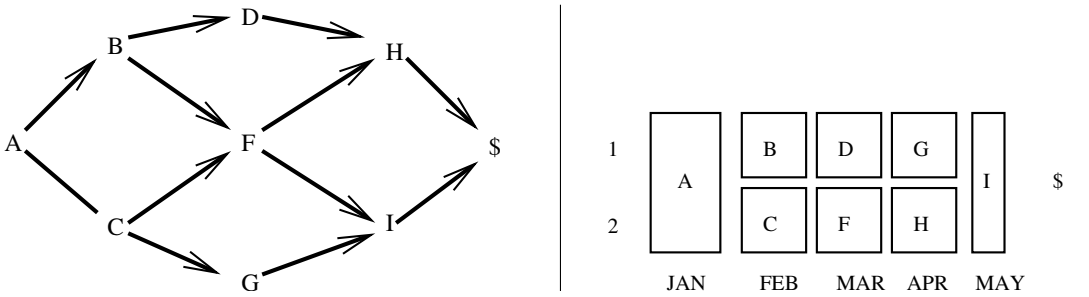

INPUT OUTPUT

14.9 Job Scheduling

Input description: A directed acyclic graph $G = (V, E)$, where vertices represent jobs and edge (u, v) implies that task u must be completed before task v.

Problem description: What schedule of tasks completes the job using the minimum amount of time or processors?

Discussion: Devising a proper schedule to satisfy a set of constraints is fundamental to many applications. Mapping tasks to processors is a critical aspect of any parallel-processing system. Poor scheduling can leave all other machines sitting idle while one bottleneck task is performed. Assigning people-to-jobs, meetings-to-rooms, or courses-to-exam periods are all different examples of scheduling problems.

Scheduling problems differ widely in the nature of the constraints that must be satisfied and the type of schedule desired. Several other catalog problems have application to various kinds of scheduling:

- Topological sorting can construct a schedule consistent with the precedence constraints. See Section 15.2 (page 481).
- Bipartite matching can assign a set of jobs to people who have the appropriate skills for them. See Section 15.6 (page 498).
- Vertex and edge coloring can assign a set of jobs to time slots such that no two interfering jobs are assigned the same time slot. See Sections 16.7 and 16.8.
- Traveling salesman can schedule select the most efficient route for a delivery person to visit a given set of locations. See Section 16.4 (page 533).

- Eulerian cycle can construct the most efficient route for a snowplow or mailman to completely traverse a given set of edges. See Section 15.7 (page 502).

Here we focus on precedence-constrained scheduling problems for directed acyclic graphs. Suppose you have broken a big job into a large number of smaller tasks. For each task, you know how long it should take (or perhaps an upper bound on how long it might take). Further, for each pair of tasks you know whether it is essential that one task be performed before the other. The fewer constraints we have to enforce, the better our schedule can be. These constraints must define a directed acyclic graph—acyclic because a cycle in the precedence constraints represents a Catch-22 situation that can never be resolved.

We are interested in several problems on these networks:

- *Critical path* – The longest path from the start vertex to the completion vertex defines the *critical path*. This can be important to know, for the only way to shorten the minimum total completion time is to reduce the time of one of the tasks on each critical path. The tasks on the critical paths can be determined in $O(n + m)$ time using the dynamic programming presented in Section 15.4 (page 489).

- *Minimum completion time* – What is the fastest we can get this job completed while respecting precedence constraints, assuming that we have an unlimited number of workers. If there were *no* precedence constraints, each task could be worked on by its own worker, and the total time would be that of the longest single task. If there are such strict precedence constraints that each task must follow the completion of its immediate predecessor, the minimum completion time would be obtained by summing up the times for each task.

  The minimum completion time for a DAG can be computed in $O(n+m)$ time. The completion time is defined by the critical path. To get such a schedule, consider the jobs in topological order, and start each job on a new processor the moment its latest prerequisite completes.

- *What is the tradeoff between the number of workers and completion time?* – What we really are interested in is how best to complete the schedule with a given number of workers. Unfortunately, this and most similar problems are NP-complete.

Any real scheduling application is likely to present combinations of constraints that will be difficult or impossible to model using these techniques. There are two reasonable ways to deal with such problems. First, we can ignore constraints until the problem reduces to one of the types that we have described here, solve it, and then see how bad it is with respect to the other constraints. Perhaps the schedule can be easily modified by hand to satisfy other esoteric constraints, like keeping Joe and Bob apart so they won't kill each other. Another approach is to formulate

your scheduling problem via linear-integer programming (see Section 13.6 (page 411)) in all its complexity. I always recommend starting out with something simple and see how it works before trying something complicated.

Another fundamental scheduling problem takes a set of jobs without precedence constraints and assigns them to identical machines to minimize the total elapsed time. Consider a copy shop with k Xerox machines and a stack of jobs to finish by the end of the day. Such tasks are called *job-shop scheduling*. They can be modeled as bin-packing (see Section 17.9 (page 595)), where each job is assigned a number equal to the number of hours it will take to complete, and each machine is represented by a bin with space equal to the number of hours in a day.

More sophisticated variations of job-shop scheduling provide each task with allowable start and required finishing times. Effective heuristics are known, based on sorting the tasks by size and finishing time. We refer the reader to the references for more information. Note that these scheduling problems become hard only when the tasks cannot be broken up onto multiple machines or interrupted (preempted) and then rescheduled. If your application allows these degrees of freedom, you should exploit them.

Implementations: JOBSHOP is a collection of C programs for job-shop scheduling created in the course of a computational study by Applegate and Cook [AC91]. They are available at *http://www2.isye.gatech.edu/∼wcook/jobshop/*.

Tablix (*http://tablix.org*) is an open source program for solving timetabling problems faced by real schools. Support for parallel/cluster computation is provided.

LEKIN is a flexible job-shop scheduling system designed for educational use [Pin02]. It supports single machine, parallel machines, flow-shop, flexible flow-shop, job-shop, and flexible job-shop scheduling, and is available at *http://www.stern.nyu.edu/om/software/lekin*.

For commercial scheduling applications, ILOG CP is reflective of the state-of-the-art. For more, see *http://www.ilog.com/products/cp/*.

Algorithm 520 [WBCS77] of the *Collected Algorithms of the ACM* is a Fortran code for multiple-resource network scheduling. It is available from Netlib (see Section 19.1.5 (page 659)).

Notes: The literature on scheduling algorithms is a vast one. Brucker [Bru07] and Pinedo [Pin02] provide comprehensive overviews of the field. The *Handbook of Scheduling* [LA04] provides an up-to-date collection of surveys on all aspects of scheduling.

A well-defined taxonomy exists covering thousands of job-shop scheduling variants, which classifies each problem $\alpha|\beta|\gamma$ according to (α) the machine environment, (β) details of processing characteristics and constraints, and (γ) the objectives to be minimized. Surveys of results include [Bru07, CPW98, LLK83, Pin02].

Gantt charts provide visual representations of job-shop scheduling solutions, where the x-axis represents time and rows represent distinct machines. The output figure above illustrates a Gantt chart. Each scheduled job is represented as a horizontal block, thus identifying its start-time, duration, and server. Project precedence-constrained scheduling

techniques are often called PERT/CPM, for *Program Evaluation and Review Technique/Critical Path Method*. Both Gantt charts and PERT/CPM appear in most textbooks on operations research, including [Pin02].

Timetabling is a term often used in discussion of classroom and related scheduling problems. PATAT (for *Practice and Theory of Automated Timetabling*) is a biannual conference reporting new results in the field. Its proceedings are available from *http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml*.

Related Problems: Topological sorting (see page 481), matching (see page 498), vertex coloring (see page 544), edge coloring (see page 548), bin packing (see page 595).