DAG scheduling

Guoli Ding and Rod Tohid

September 19, 2019

The inputs.

- A positive integer m, indicating that there are m machines available.
- An acyclic directed graph $D = (V, E)$, known as a DAG. Here a vertex represents a job and a directed edge $e = uv$ indicates that job v cannot start before job u finishes.
- A nonnegative weight function ℓ on $V \cup E$, where ℓ_v indicates the amount of time required to finish job v, and ℓ_{uv} indicates the amount of time required for v to access u if u, v are located on different machines.

Assumptions.

- A machine can process at most one job at a time.
- If a machine starts a job then it has to finish the job before it can start another job. In particular, this implies that a job can only be completed by one machine.
- For each job v, the weight ℓ_v is the same on every machine. That is, a job can be done by any machine with the same amount of time, or simply put, the machines are identical.
- For any edge $e = uv$, the weight ℓ_e remains the same no matter which two machines process u and v. In other words, the communication channels are identical.

The problem.

- Decide which job goes to which machine.
- Decide a starting time for each job.
- Minimize the final completion time of all jobs.

Linear programming formulation. Let $M_1, ..., M_m$ denote the machines and let $v_1, ..., v_n$ denote the jobs. We introduce the following variables.

- t_i: the starting time for v_i
- T: final finishing time
- x_{ij}: the assignment indicator, which is 1 if job v_i is assigned to machine M_j, and is 0 if otherwise
- y_{ij}: the time line indicator, which is 1 if $t_i \le t_j$ and is 0 if $t_i > t_j$
- L : defined as the sum of ℓ over $V \cup E$, which is the total amount of involved time

The following is the formulation.

$$\min\ T$$

subject to:

$$x_{i1} + x_{i2} + \cdots + x_{im} = 1, \quad \forall v_i \in V \tag{1}$$

$$t_j - t_i - \ell_i \ge \ell_{ij} \max\{x_{ik} - x_{jk} : k = 1, ..., m\}, \quad \forall v_i v_j \in E \tag{2}$$

$$t_j - t_i - \ell_i \ge L(\max\{x_{ik} + x_{jk} - 2 : k = 1, ..., m\} + y_{ij} - 1), \quad \forall v_i, v_j \in V \text{ with } i < j \tag{3a}$$

$$t_i - t_j - \ell_j \ge L(\max\{x_{ik} + x_{jk} - 2 : k = 1, ..., m\} - y_{ij}), \quad \forall v_i, v_j \in V \text{ with } i < j \tag{3b}$$

$$T \ge \max\{t_i + \ell_i : i = 1, ..., n\} \tag{4}$$

$$t_i \ge 0; \ \ x_{ij} \in \{0, 1\}; \ y_{ij} \in \{0, 1\}$$

Note that the number of variables is $n + 1 + mn + \binom{n}{2}$ and the umber of inequalities is $m(n^2 + |E| - n) + 3n$. Some of the inequalities/variables are redundant and thus can be removed.

Greedy Algorithm.

- Keep tract of the set R of ready jobs (with cleared dependencies)
- As soon as a machine becomes available,
  - update R
  - assign a job from R to this machine
- Repeat until all jobs are scheduled.

REMARKS.

- At any point in time, let F denote the set of jobs that are finished (so $F = \emptyset$ in the very beginning). Then R consists of jobs v such that no edge of $D \backslash F$ is directed to v. That is, the in-degree of v is zero in $D \backslash F$.
- When assigning $v \in R$ to an available machine, there are many choices. We could develop different principles for making such a choice and let experiments decide which is better –this is further research.

An application. Suppose we have a user program $\mathbb{P}$, by which we mean a sequence of matrix operations (an example of $\mathbb{P}$ is illustrated below). Program $\mathbb{P}$ can be expressed by a computation graph $\mathbb{G}$ (which is essentially the execution tree) and this graph is a DAG (as illustrated below). If we want to process $\mathbb{P}$ in m machine then we have exactly the DAG scheduling problem.