Scheduling

Timothy M Schwieg

3 March 2018

ROADMAP OF SEMINAR

1. Introduction to Scheduling
2. Parts of each algorithm
3. Earliest Due Date
4. Shortest Processing Time
5. Moore's Algorithm
6. Preemption costs

History and Focus

- 1874: Frederick Taylor and Henry Gatt coin the terms scientific management. Make public schedules but do not solve any problems
- 1954: Selmer Johnson while working for RAND publishes a solution for solving a two-machine process of book binding.
- Focus upon a single machine. However the order we do it is irrelevant to the time taken to solve the problem without order constraints.

Complications

- Differing weights between tasks - usually managable
- Release times - Destroys Tractibility
- Precendence Constraints
- Preemption
- Online Algorithms

Earliest Due Date - Minimize Maximum Lateness

- Cost Function: $c_j(t) = (t - d_j)^+$
- Find a job k such that:
  $c_k(T) = \min\{(T - d_j)^+\} T = \sum_{n=0}^{N} p_n$
- This is equivalent to taking the task with the maximum due date.
- Complexity is $\mathcal{O}(N \log(N))$, Independent of Processing Time.

Precedence Constraints

- We can't pick any job to go last, only since some jobs must precede others. Define J to be the sets of jobs that do not have to precede others.
- Find a job $k \in J$ where $c_k(T) = \min_{j \in J}\{(T - d_j)^+\}$
- Schedule that job k to be last, recalculate the set J to be the jobs not including k that can go last and repeat.
- This is $\mathcal{O}(N^2)$ and processing time does not matter

Preemption

- Machine can switch to tasks for free, so only time machine is idle is when nothing can be done
- Seek blocks where there is no downtime.
- Sort by release dates, and use this to break up the tasks into blocks. This can be done in $\mathcal{O}(N)$ time.
- Find a job l in each block where $c_l(t(B)) = \min_{b \in B}\{(t(B) - d_b)^+\}$
- Schedule this job last, then continue scheduling the block with the same policy until it is fully scheduled.
- This runs in $\mathcal{O}(N^2)$ runtime, completing at most N-1 preemptions

Shortest Processing Time - Minimizing Outstanding Tasks

- We want to get things done as quickly as possible, so start with the tasks we can do the quickest
- Sort the tasks based on processing time, and complete in ascending order
- This can be computed in $\mathcal{O}(N \log(N))$ run-time as it only really requires a sort.

Complications

- Weights:
  - Greedy Algorithm pays off - we can sort it based on: $\frac{p_j}{w_j}$.
  - this is still $\mathcal{O}(N \log(N))$
- Pre-emption:
  - Switch at a start time when a new task is available, if it has a lower processing time, otherwise continue as before.
- Precedence Relations:
  - This becomes NP-hard, and it is pretty difficult to break the computational problem. Online algorithm is one method.

Online Alogrithm

- Greedy Algorithm doesn't work here, as we can be stuck doing something when it is not optimal.
- Change suggested by: Anderson and Potts 2004 - When a new job appears, compare it to the current job by minimizing $\frac{p_j}{w_j}$, and scheduling job j to start at: $\max\{t, p_j\}$.
- This ensures the algorithm is 2-competitive, at worst case twice as bad as the offline version. This is the best any online algorithm can do.

Moore's Algorithm - Minimizing Number of Late Tasks

- No penalty for how late a task is - No longer seeking a permutation, only an optimal subset of the tasks
- This leaves: $\sum_{n=0}^{N} \binom{N}{n} = 2^N$ possible partitions.
- Want to use Dynamic Programming to turn this into $\mathcal{O}(N \log(N))$ or $\mathcal{O}(N^2)$.
- Integer Program: x_n denotes if job n, ordered by earliest due date, is late or not.
- $\max \sum_{n=1}^{N} w_n x_n$ subject to: $\sum_{i=1}^{j} p_i x_i \leq d_j \quad \forall j 1 \leq j \leq n$

Dynamic Programming Algorithm

- Create a function $F_j(t)$ which tells us the optimal subset of first j jobs to complete in time t.
- We seek $F_N(T)$ where T is the time required to do all the jobs we choose to do.
- Base Case: $F_0(0) = 0, F_0(t) = \infty, t > 0$.
- If we add a new job: $F_j(t) = F_{j-1}(t) + w_j$
- Don't add job: $F_j(t) = F_{j-1}(t - p_j)$

Dynamic Programing Algorithm Conintued

- Combine these two into a decision rule:

$$F_j(t) = \begin{cases} \min\{F_{j-1}(t - p_j), F_{j-1}(t) + w_j\} & \text{if } 0 \leq t \leq d_j \\ F_j(d_j) & \text{if } d_j \leq t \leq T \end{cases}$$

- Algorithm: For j in 1:N do:
  for t in 0:T do:
  Update $F_j(t)$
- Optimal value is then $F_n(d_n)$ in $\mathcal{O}(N^2)$ time.

Complications

- Symmetry of the constraints was used to build the program, not possible once precedence constraints and release dates are input.
- This returns to the integer programs we saw in Operations Research. These problems are NP-Hard.
- If preemption is allowed, the problem may be solved in Pseudo-Polynomial time. See Lawler (1990) for a $\mathcal{O}(NK^2W^2)$ algorithm.

Preemption Costs

- It is not always costs less to switch between tasks, but this is difficult to compute tractably. This price is called a context-switch.
- The time spent switching between tasks is called meta-work.
- This creates a trade-off between Responsiveness, how quickly a machine changes tasks, and Throughput, how much a machine can accomplish.
- This is a vector optimization problem, and choices on the Pareto Frontier must be made in the context of the specific machine. E.x. Computers limit responsiveness of the mouse to the smallest interval the eye can detect, but may increase it during a video game.

Summary and Conclusions

- What objective you choose determines what order tasks should be done, and is dependent on your needs for the machine
- Adding release dates or precedence constraints often has the algorithm exit the realm of tractable problems.
- Online algorithms and Preemption allow for a return back to Tractibility.
- Preemption is not always free, and context-switching too often can lower throughput, or cause thrashing at worst case.