Network QoS
371-2-0213

Lecture 8

Gabriel Scalosub

Outline

Scheduling Preliminaries

- Over 50 years of research
- Model:
  - n jobs J_j, $j = 1, \ldots, n$
  - m machines M_i, $i = 1, \ldots, m$
  - p_{ij}: job J_j's processing time on machine M_i
  - jobs should be scheduled on machines
    - various constraints/assumptions
    - objective function is optimized
- Examples:
  - schedule jobs on m machines
    - variable-length jobs (same on all machines)
    - goal: minimize maximum completion time (Makespan)
    - single machine: trivial...
    - two machines: NP-hard (subset-sum)

Machine Scheduling Notation: $\alpha \mid \beta \mid \gamma$

- Machine environment α:
  - m: number of machines
  - P: parallel identical machines
  - Q: parallel machines with speeds, $p_{ij} = p_j / s_i$
    - job processing requirement p_j, machine speed s_i (independent)
  - R: parallel unrelated machines
  - ...
- Job characteristics β:
  - r_j: release time
  - d_j: deadline
  - pmtn: preemption is allowed
    - a job assigned to a machine can be stopped, and rescheduled later
  - prec: precedence constraints
    - specified by an underlying DAG on the jobs
  - ...

Machine Scheduling Notation: $\alpha \mid \beta \mid \gamma$ (Cont.)

- Objective function γ:
  - $C_{\max}$: minimize maximum completion time of a job
    - job J_j's completion time C_j
  - $L_{\max}$: minimize maximum lateness of a job
    - job J_j's lateness $L_j = C_j - d_j$
  - $\sum_j U_j$: minimize number of late jobs
    - job J_j's lateness indicator $U_j = \mathbb{1}_{C_j > d_j}$
  - $\sum_j w_j \overline{U}_j$: maximize weight of jobs scheduled by their deadlines
    - job J_j's non-lateness indicator $\overline{U}_j = 1 - U_j$
  - ...
- Examples:
  - $1 \mid d_j, r_j \mid \sum_j w_j \overline{U}_j$
    - one machine ("queue"), maximize weight of jobs scheduled by their deadlines, with release times.
    - similar to the problem we saw last week...
  - $Q2 \mid \mathrm{pmtn}, \mathrm{prec}, d_j \mid \sum_j L_j$
    - 2 machines (different speeds), preemption allowed, precedence constraints, minimize sum (equiv. – average) of latenesses

Machine Scheduling Notation: $\alpha \mid \beta \mid \gamma$ (Cont.)

- One more example:
  - $Pm \mid\mid C_{\max}$
    - m identical machines, minimize the *makespan*
    - coming up next...

Examples of Problems, or "The Scheduling Zoo"

http://www.informatik.uni-osnabrueck.de/knust/class/,
http://schedulingzoo.lip6.fr/

Parallel machine problems without preemption

- **maximal polynomially solvable:**

$P\|p_i = p; outtree; r_i\|C_{max}$	Brucker et al. (1977)
$P\|p_i = p; tree\|C_{max}$	Hu (1961), Davida & Linton (1976)
$Q\|p_i = p; r_i\|C_{max}$	Assignment-problem
$Q2\|p_i = p; chains\|C_{max}$	Brucker et al. (1999)
$P\|p_i = 1; chains; r_i\|L_{max}$	Dror et al. (1998), Baptiste et al. (2004)
$P\|p_i = p; intree\|L_{max}$	Brucker et al. (1977), Monma (1982)
$P2\|p_i = p; prec\|L_{max}$	Garey & Johnson (1976)
$P2\|p_i = 1; prec; r_i\|L_{max}$	Garey & Johnson (1977)
$P\|p_i = 1; outtree; r_i\|\sum C_i$	Brucker et al. (2002), Huo & Leung (2005)
$P\|p_i = p; outtree\|\sum C_i$	Hu (1961)
$P2\|p_i = 1; prec; r_i\|\sum C_i$	Baptiste & Timkovsky (2004)
$P2\|p_i = p; prec\|\sum C_i$	Coffman & Graham (1972)
$Pm\|p_i = p; tree\|\sum C_i$	Baptiste et al. (2004)
$Qm\|p_i = p; r_i\|\sum C_i$	Dessouky et al. (1990)
$R\|\|\sum C_i$	Horn (1973), Bruno et al. (1974)
$P\|p_i = p; r_i\|\sum w_i C_i$	Brucker & Kravchenko (2008)
$P\|p_i = 1; r_i\|\sum w_i U_i$	Networkflowproblem
$Pm\|p_i = p; r_i\|\sum w_i U_i$	Baptiste et al. (2004)
$Q\|p_i = p\|\sum w_i U_i$	Assignment-problem
$P\|p_i = p; r_i\|\sum T_i$	Brucker & Kravchenko (2005)
$P\|p_i = 1; r_i\|\sum w_i T_i$	Networkflowproblem
$Q\|p_i = p\|\sum w_i T_i$	Assignment-problem

- **maximal pseudopolynomially solvable:**

$Qm\|r_i\|C_{max}$	Lawler et al. (1989)
$Qm\|\|\sum w_i C_i$	Lawler et al. (1989)
$Qm\|\|\sum w_i U_i$	Lawler et al. (1989)

- **minimal NP-hard:**

	$P2\|\|C_{max}$	Lenstra et al. (1977)
*	$P\|\|C_{max}$	Garey & Johnson (1978)
*	$P\|p_i = 1; intree; r_i\|C_{max}$	Brucker et al. (1977)
*	$P\|p_i = 1; prec\|C_{max}$	Ullman (1975)
*	$P2\|chains\|C_{max}$	Du et al. (1991)
*	$Q\|p_i = p; chains\|C_{max}$	Kubiak (1988)
*	$P\|p_i = 1; outtree\|L_{max}$	Brucker et al. (1977)
*	$P\|p_i = 1; intree; r_i\|\sum C_i$	Lenstra (-)
*	$P\|p_i = 1; prec\|\sum C_i$	Lenstra & Rinnooy Kan (1978)

Graham's List Scheduling for Minimizing Makespan ('66)

Algorithm 1 LS

sort jobs arbitrarily, $J_1, \ldots, J_n$
for $j = 1, \ldots, n$ **do**
    assign J_j to currently least loaded machine
end for

- Probably the first approximation algorithm ever
- Actually an *online* algorithm
  - since order is arbitrary
- How bad can LS be?
  - Assume job order is (considering processing times)

$$\underbrace{1, 1, \ldots, 1}_{m(m-1) \text{ jobs}}, m$$

  - $\frac{\mathrm{LS}}{\mathrm{OPT}} = \frac{2m-1}{m} = 2 - \frac{1}{m}$

Competitive Analysis of List Scheduling

Theorem

Algorithm LS *is* $(2-\frac{1}{m})$*-approximation (competitive)*

Proof.

- Observation: $\textsc{Opt} \geq \max\left\{\frac{\sum_j p_j}{m}, \max_j p_j\right\}$
- Notation:
  - L: maximum load in LS
  - i^*: most loaded machine in LS
  - j^*: last job assigned to i^*
- All machines have load $\geq L - p_{j^*}$

$\Rightarrow$ $(\sum_j p_j) - p_{j^*} \geq m(L - p_{j^*})$

$$\begin{aligned} L &\leq \frac{(\sum_j p_j) - p_{j^*}}{m} + p_{j^*} \\ &= \frac{\sum_j p_j}{m} + (1-\frac{1}{m})p_{j^*} \\ &\leq (2-\frac{1}{m})\textsc{Opt} \end{aligned}$$

□

Improving upon List Scheduling

- Observations:
  - if $n \leq m$, LS is optimal
  - ensuring load-balancing online might be harmful
- How would improved online algorithms look like?
  - purposely cause imbalance
    - prepare for the worse
    - leave "some" room for a big job
  - not too much

	deterministic			randomized	
m	lower bound	upper bound	LS	lower bound	upper bound
2	1.5000	1.5000	1.5000	1.3333	1.3334
3	1.6666	1.6667	1.6667	1.4210	1.5567
4	1.7310	1.7333	1.7500	1.4628	1.6589
5	1.7462	1.7708	1.8000	1.4873	1.7338
6	1.7730	1.8000	1.8333	1.5035	1.7829
7	1.7910	1.8229	1.8571	1.5149	1.8169
∞	1.8520	1.9230	2.0000	1.5819	–

Improving upon List Scheduling

- What if we're allowed to sort jobs?
  - no longer online...
  - can we get a better approximation guarantee?
- $\textsc{OrderedLS}$: jobs are sorted: $p_1 \geq p_2 \geq \ldots \geq p_n$

Lemma

If $n > m$ then $\textsc{Opt} \geq p_m + p_{m+1}$

Proof.

- Pigeonhole principle (for any solution):
  - 2 jobs out of $J_1, \ldots, J_{m+1}$ are assigned to same machine
  - jobs are ordered by non-increasing processing time
  - the minimum weight of any such 2 jobs is $p_m + p_{m+1}$

$\square$

Analysis of Improved Offline List Scheduling

Theorem

Algorithm $\textsc{OrderedLS}$ *is a* $(\frac{3}{2} - \frac{1}{2m})$*-approximation*

Proof (similar to previous).

- Observation: $\textsc{Opt} \geq \max\left\{\frac{\sum_j p_j}{m}, \max_j p_j\right\}$
- Same notation:
  - L (max-load), i^* (max-load machine), j^* (last job in i^*)
- If $j^* \leq m$: $\textsc{OrderedLS} = \textsc{Opt}$
- If $j^* > m$: $p_{j^*} \leq \frac{p_m + p_{m+1}}{2} \leq \frac{\textsc{Opt}}{2}$

$$\begin{aligned} L &\leq \frac{(\sum_j p_j) - p_{j^*}}{m} + p_{j^*} \\ &= \frac{\sum_j p_j}{m} + (1 - \frac{1}{m})p_{j^*} \\ &\leq \textsc{Opt} + (1 - \frac{1}{m})\frac{\textsc{Opt}}{2} \\ &\leq (\frac{3}{2} - \frac{1}{2m})\textsc{Opt} \end{aligned}$$

□

Interval Scheduling

- Input
  - job J_j requires processing during interval $I_j = [s_j, t_j)$
    - uses *all* the interval
    - each job J_j has weight $w(J_j)$
  - m identical machines
- Goal
  - find a subset of jobs $S = S_1 \dot\cup S_2 \dot\cup \ldots \dot\cup S_m$, such that
    - for any k, and any $J_j, J_{j'} \in S_k$, $I_j \cap I_{j'} = \emptyset$
  - criteria
    - $\max \sum_{J_j \in S} w(J_j)$
- Usually referred to as *interval scheduling*
  - refer to J_j as its interval
- Our focus:
  - uniform weights, i.e., $w(J_j) = 1$ for all j
- What algorithm would you use?

Offline Interval Scheduling ($m = 1$)

Algorithm 2 $\mathrm{GIS}(\text{machine}, \text{jobs } \mathcal{J})$

sort intervals in $\mathcal{J}$ ▷ How?
for $j = 1, \ldots, n$ **do**
    if J_j doesn't overlap any currently scheduled interval **then**
        add J_j
    end if
end for

- Greedy interval scheduling (GIS...)
- Main question: how should we sort intervals?

Offline Interval Scheduling ($m = 1$)

Theorem

GIS *(sorting by t_j) produces an optimal schedule*

Proof (by contradiction).

- Let $J_{j_1}, \ldots J_{j_k}$ be the schedule produced by GIS
- Let OPT be an optimal schedule
    - OPT schedules $J_{j_1}, \ldots, J_{j_r}, J_{j^*_{r+1}}, \ldots$
    - $r \leq k$ is maximal
    - i.e., OPT "agrees" with GIS on a maximum prefix of GIS
- if $r = k$, GIS would have scheduled additional jobs
    - at least $J_{j^*_{r+1}}$ is available and non-overlapping
- Assume $r < k$
    - by definition of GIS, $t_{j_{r+1}} \leq t_{j^*_{r+1}}$
    - we can switch them without harming OPT
- contradiction to GIS definition / maximality of r □

Offline Interval Scheduling ($m \geq 1$)

Algorithm 3 $m\text{-}\mathrm{GIS}(m \text{ machines, jobs } \mathcal{J})$

1: **for** $i = 1, \ldots, m$ **do**
2: add $\mathrm{GIS}(\text{machine } i, \text{remaining jobs})$ ▷ GIS_i
3: **end for**

Theorem

m-GIS is a 2-approximation

Proof (by contradiction).

- $\textsc{Opt}_i$: set of intervals in $\textsc{Opt} \setminus m\text{-}\mathrm{GIS}$ scheduled on M_i
- Assume $|m\text{-}\mathrm{GIS}| < |\textsc{Opt}|/2$

  $\Rightarrow |\cup_i \textsc{Opt}_i| > |\textsc{Opt}|/2 > |m\text{-}\mathrm{GIS}| = |\cup_i \mathrm{GIS}_i|$
- Pigeonhole principle
  - exists i s.t. $|\textsc{Opt}_i| > |\mathrm{GIS}_i|$
  - $\textsc{Opt}_i$ is available to GIS_i: contradicts optimality of GIS □

Online Interval Scheduling

- Job J_j arrives at s_j
  - reveals t_j
- preemptive
  - otherwise, same as ordering by s_j (unbounded competitiveness)

Algorithm 4 $\textsc{Online-}m\textsc{-GIS}$: at time t

```
for any job J_j arriving at t do
    if J_j can be scheduled on some machine M_i then
        schedule J_j on M_i
    else
        j' ← arg max_k {t_k | J_k is scheduled and intersects J_j}
        if t_j ≥ t_j' then drop J_j
        else drop J_j' and schedule J_j instead
        end if
    end if
end for
```

Online Interval Scheduling

- Online-m-GIS emulates m-GIS

Corollary

Online-m-GIS *is 2-competitive*

- Is this tight?
- If m-GIS is an α-approximation
  - Online-m-GIS is α-competitive

Theorem

m-GIS *produces an optimal schedule*

Corollary

Online-m-GIS *produces an optimal schedule*

- Online!!

From Networking to Scheduling

- Links / paths $\longrightarrow$ machines
- Traffic $\longrightarrow$ jobs
- Applications:
  - routing
  - optical networks
    - Wavelength Division Multiplexing (WDM)
  - packet scheduling
    - bounded-delay model
    - forwarding across networks (sensors, bounded-buffers)
  - many more...

- Some references

  - Cormen, Leiserson, Rivest. Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.
  - Sgall, Online Scheduling - A Survey, in Fiat et al (eds.), Online Algorithms: The State of the Art, 1998.
  - Graham et al., Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, 1979.

The Creative Part of the Course

- Models we've seen:
  - AAP Routing
  - BM with commitments (FIFO)
  - BM with packet weights (FIFO)
  - BM/Scheduling with packet weights and deadlines
- Issues to take into account:
  - interesting/useful model extension
  - better algorithms
    - proofs (analytic)
    - simulations (heuristic)
  - KISS: "Keep It Simple, Stup..."
- Analytic approach:
  - toy examples are usually very instructive
  - limited models are a good place to start
- Simulation approach:
  - try to highlight the main ideas/benefits
  - a great motivator for designing better solutions