Minimizing flow time on unrelated machines

Nikhil Bansal

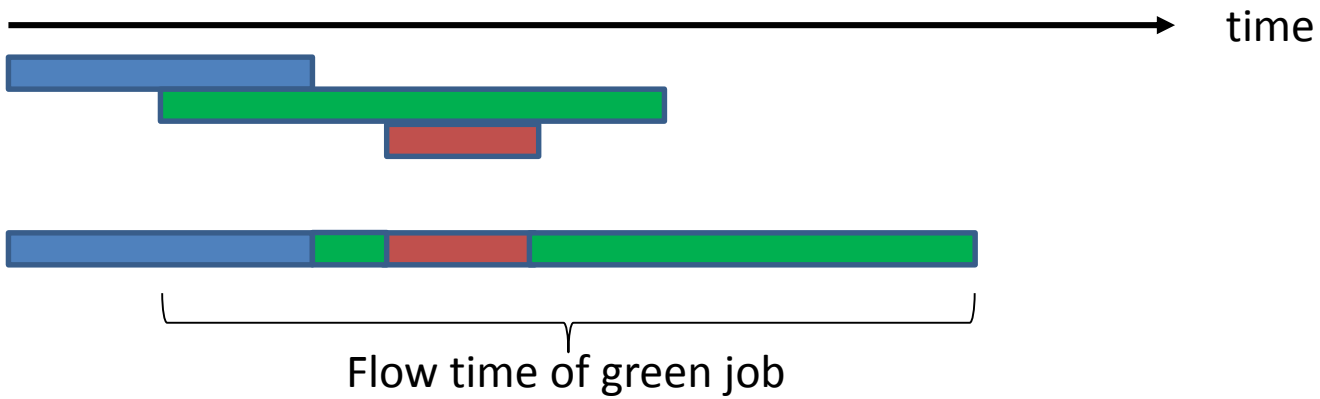
Janardhan Kulkarni (MSR Redmond)

Flow time

Classic single machine setting

Jobs: size p_j, release time r_j

Flow time: $f_j = C_j - r_j$ (completion time – release time)



Objective: Minimize total flow time (SRPT is optimal)
Minimize max. flow time (FIFO is optimal)

Total Flow time

Optimum algorithm: Shortest Remaining Processing Time

Work on the job with least remaining processing.

Useful fact: Total flow time $= \sum_j f_j = \sum_t n(t)$

(n(t) = number of unfinished jobs at time t.)

Proof: Suppose each job pays \$1 at each time step it is alive.

Job j's payment = f_j So, total payment = $\sum_j f_j$

Total payment at time t = n(t) So, total payment = $\sum_t n(t)$

Multiple machines

Machines 1,…m Jobs 1,…,n

p_{ij}: Size of job j on machine i

Identical machines

Related machines: speed s_i, $p_{ij} = p_j / s_i$

Restricted assignment: $p_{ij} \in \{ p_j, \infty \}$

…

Unrelated: p_{ij} arbitrary

No Migration: Job must be processed on a single machine

What is known?

Identical : O(log P) [Leonardi and Raz'96] + follow ups

Ω(log P) hardness [Garg-Kumar 06] (P: max/min job size)

O(log P) for related and restricted case [2005-2008]

(clever, LP based techniques)

Unrelated case: O(k) if k distinct p_{ij} values [Garg-Kumar 08, Sitters 08]

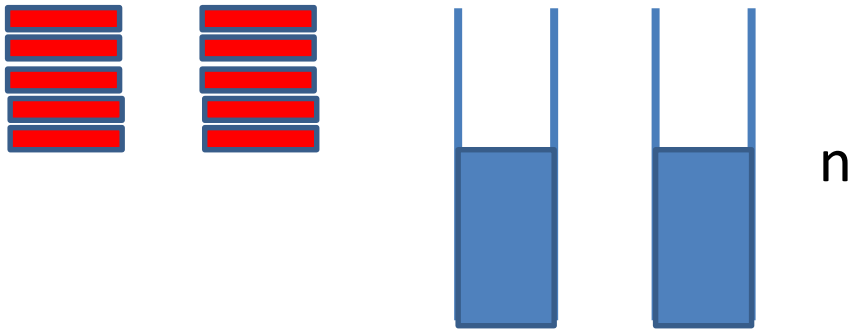
Perhaps round to powers of 2 (or $1+\epsilon$) ?

Flow time is very sensitive to sizes (never mess with them)

Case 1: A Job of size 1, released at t=0,1,2,…,n O(n) total flow

Case 2: A job of size 1.1, released at t=0,1,2,…,n $\Omega(n^2)$ total flow

Another Instructive example



Instance: m=2 machines. All job sizes = 1.

2n jobs released t = 0. Two jobs each at t = n, n+1, …, T

If imbalance in blue jobs (e.g. random assignment)

$\sqrt{n}$ imbalance -> $\Omega(\sqrt{n})$ approximation

(OPT $\approx$ 2T, Alg $\approx \sqrt{n}$ T)

Our Result

Thm: An O($\log n \log P$) approx. for total flow on unrelated machines

Remark: Can be made O($\log^2 n$)

Thm: An O(log n) approx. for maximum flow on unrelated machines

Previously: $(1+\epsilon, O(1/\epsilon))$ approximation [Garg et al 2013]

Technique: Iterated Rounding

Outline

- Introduction
- Iterated Rounding

  (Lenstra, Shmoys, Tardos: makespan on unrelated machines)

  Note: Same as max-flow time if all release times = 0
- Total flow time

Iterated Rounding

LP max cx

$Ax \leq b$

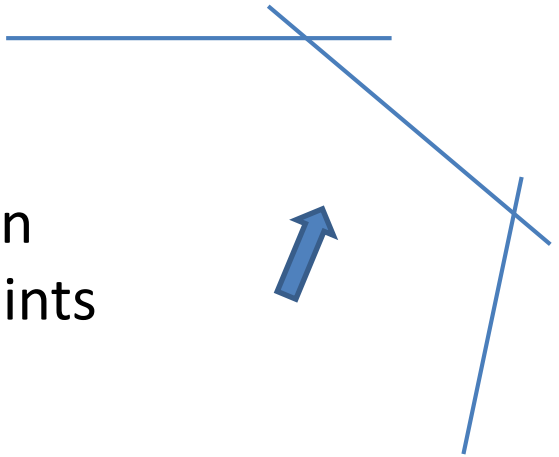
$0 \leq x \leq 1$

n variables

m non-trivial constraints

Obs: There is an optimum solution with $\geq$ n-m 0-1 variables.

Vertex: determined by solution of n (tight) linearly independent constraints



Iterated Rounding

LP max cx

$Ax \leq b$

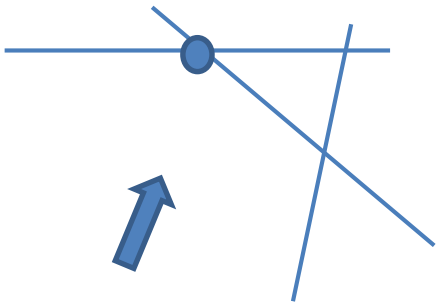
$0 \leq x \leq 1$

n > m variables (say m = n-1)

$x_1, \ldots, x_n$

Key point: If drop another constraint, get another integral variable

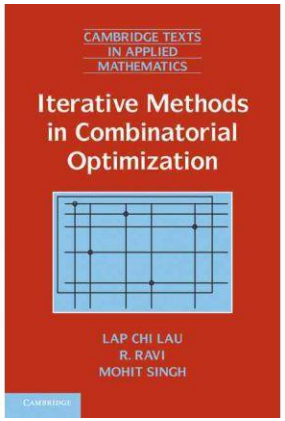
Note: objective can only go up during the iterations



Hope: Dropping a constraint does not mess up things much

Iterated Rounding

Very powerful



Karmarkar Karp'82 (bin packing)
Jain'98 (survivable network design)
Singh Lau'06 (deg bdd spanning trees)
…

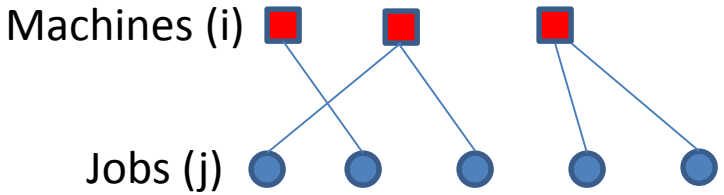
Refined variant (can "control error" on dropped constraints)
Bansal'10, Lovett-Meka'12 (via discrepancy theory)

Rothvoss'13 (bin packing)
Bansal et al'14 (broadcast scheduling)

Makespan Minimization Example

Makespan on Unrelated Machines

Target T: Want to assign jobs so that max load <= T



Thm (Lenstra, Shmoys, Tardos'90): 2-approximation (in fact T + p_{max})

Preprocess: $p_{ij} \leq T$ (if $p_{ij} > T$, set $p_{ij} = \infty$)

LP: $\sum_j p_{ij}\, x_{ij} \leq T$ for each machine i = 1,…,m

$\sum_i x_{ij} = 1$ for each job j = 1,…,n

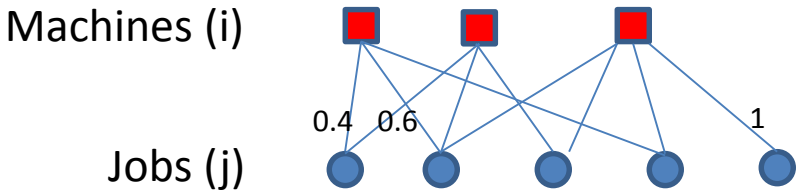
(n+m) non-trivial constraints (plus constraints $x_{ij} \geq 0$)

Makespan on Unrelated Machines

LP: $\sum_j p_{ij}\, x_{ij} \leq T$ for each machine i = 1,…,m

$\sum_i x_{ij} = 1$ for each job j = 1,…,n

(n+m) non-trivial constraints (plus constraints $x_{ij} \geq 0$)



Fix x_{ij} if 0 or 1

Fractional: $0 < x_{ij} < 1$

Claim: There is some machine i with

1) At most one fractional x_{ij} assigned to it.
2) At most two fractional x_{ij} and $x_{ij'}$, with $x_{ij} + x_{ij'} \geq 1$

Suffices: Load increase $\leq T(1 - x_{ij}) + T(1 - x_{ij'}) \leq T$.

Makespan on Unrelated Machines

LP: $\sum_j p_{ij}\, x_{ij} \leq T$ for each i = 1,…,m

$\sum_i x_{ij} = 1$ for each j = 1,…,n (plus constraints $x_{ij} \geq 0$)

Claim: There is some machine i with

1) At most one fractional x_{ij} assigned to it.
2) At most two fractional x_{ij} and $x_{ij'}$, with $x_{ij} + x_{ij'} \geq 1$

non-zero variables $\leq$ n+m Let b := non-integral jobs

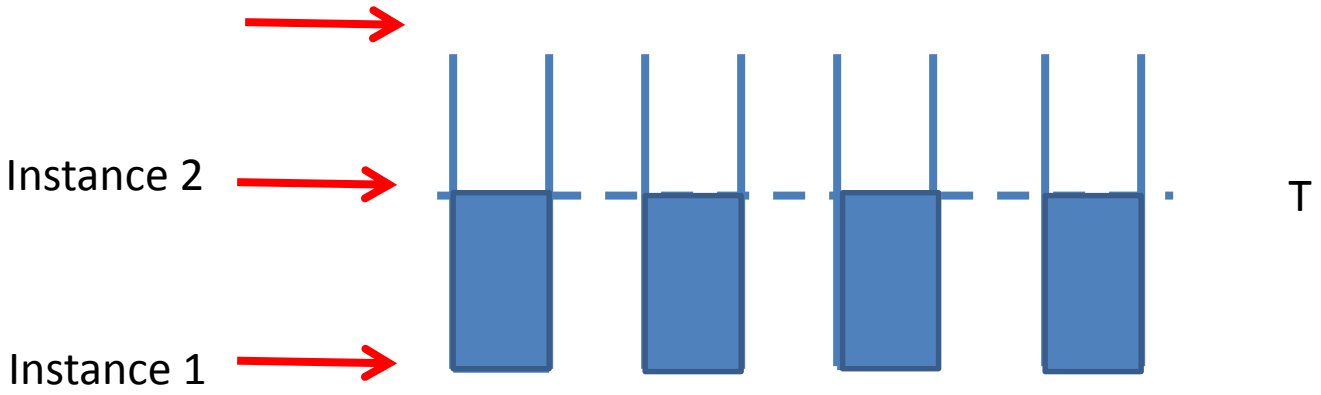
(either $x_{ij} = 1$ or $0 < x_{ij} < 1$)

2b $\leq$ # frac. variables $\leq (n+m) - (n-b) \leq m + b$

If b<m ($\leq$ 2m-1 frac. vars: so some machine <=1 variable)

If b=m. (2m frac. vars, m jobs, each machines has 2 variables) QED.

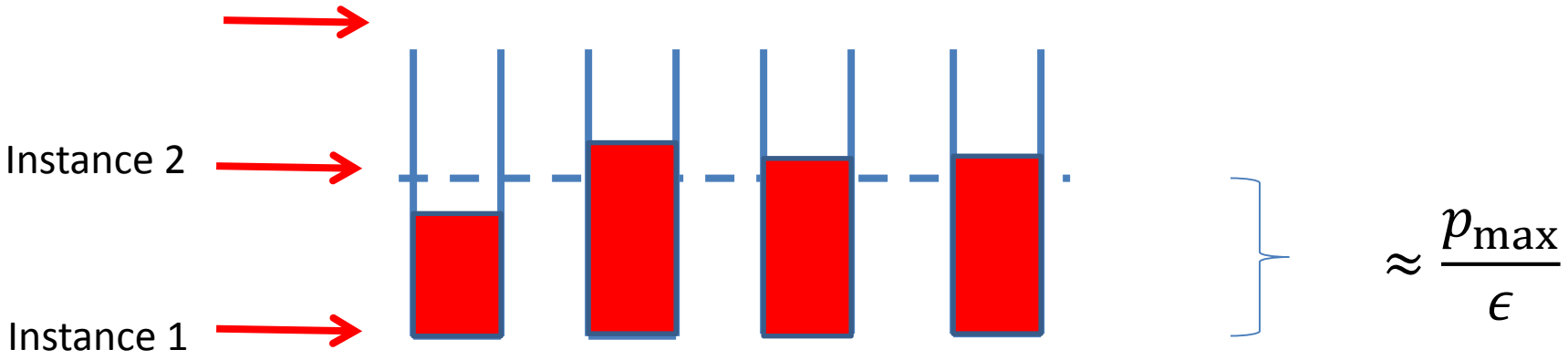
Extending to Max flow time



Instructive example:

Several Makespan instances released over time

Trouble for max flow time



Error can build up

An $(1+\epsilon, O(1/\epsilon))$ approximation [Garg et al 2013]

Outline

- Introduction
- Iterated Rounding

  (Lenstra, Shmoys, Tardos: makespan on unrelated machines)
- Total flow time

Previous Approaches

Time-indexed LP formulation

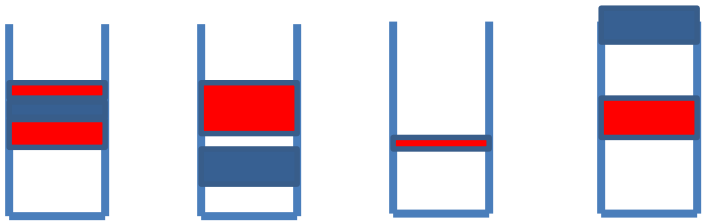
x_{ijt} : How much of job j scheduled on machine i at time t

All the constraints:

(No overload) $\sum_j x_{ijt} \leq 1$ for all (i,t)

(each job scheduled) $\sum_i ((\sum_t x_{ijt})/p_{ij}) \geq 1$ for all j

Some objective for "fractional" flow time.



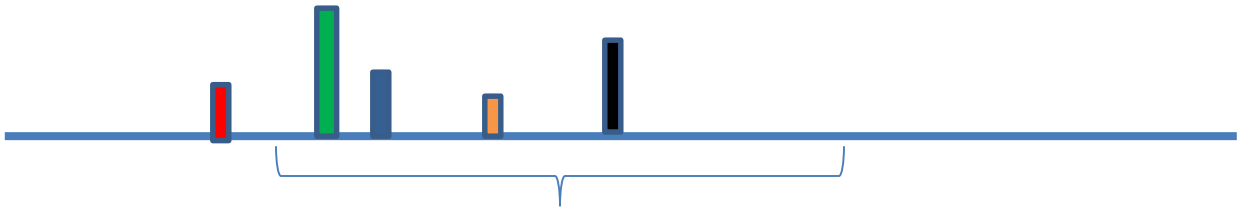
(note: LP can put a job on many machines, and schedule the pieces in parallel)

Main Idea

Reduce splitting at expense of "local overloading" $(\sum_j x_{ijt} \leq 1)$

Want 1 piece per job (i.e. $x_{ijt} = p_{ij}$)

Bounded overload: For any interval X of time (s,t)
Total work assigned to X $\leq$ (t-s) + B.



Pseudo-schedule -> Schedule: FIFO, no job delayed more than B.

Overview

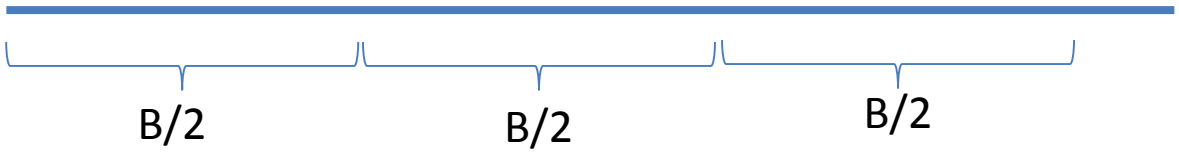
Replace $\sum_{jt} x_{ijt} \leq 1$ by "Load $\leq B$ for every interval"

At least half of the jobs assigned integrally

Repeat for O(log n) iterations.

Overload in any interval = O(B log n)

Pseudo-schedule



Form intervals of size B/2. Require $\leq$B/2 work assigned to each interval.

How small can we choose B?

Seems like >= P (need to place big job somewhere)

But, this can mess up small jobs

(E.g. B jobs of size 1 arrive at once vs. one job at t=0,1…B-1)

Have different scales: Different bounds for each size class.

Large jobs can tolerate larger B (do SRPT instead of FIFO)

LP for pseudo-schedule

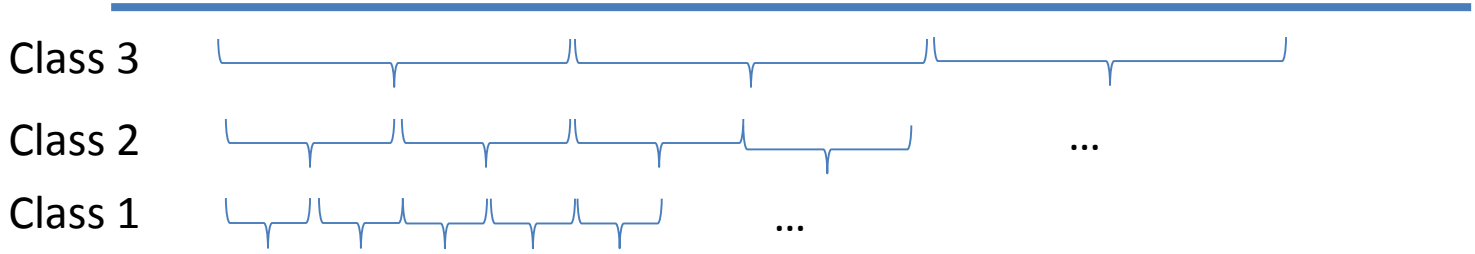
Fix machine i

Say job j has class k on machine i if $p_{ij} \in [2^{k-1}, 2^k)$

For k=1,2,…,log P consider intervals of size $10 \cdot 2^k$

LP: For each interval X of class k,

Total work of $\text{class}_{\leq k}$ jobs in X $\leq 10 \cdot 2^k$.



The key counting lemma

Number of constraints for class k.

mT/(10) 2^k interval constraints (T: time horizon)
n jobs (class <= k)

On the other hand, $n \geq mT/2^k$
non-zero vars. ≤ 1.1 n (n job constraints + n/10 interval constraints)

$2f + (n-f) \leq$ # non-zero vars. ≤ 1.1 n
$f \leq 0.1$ n
(a constant fraction of jobs assigned integrally)

Finishing off

Fix integral jobs, redefine intervals, repeat

After O(log n) iterations all jobs assigned integrally

For any interval X = (s,t)

Total work of class <= k jobs assigned to X $\leq$ |X| + O(log n $\cdot$ 2^k)

Implies $\approx$ O(log n) additional jobs pending per class

Total cost $\leq$ OPT + O(log n log P) (mT)

QED

Concluding Remarks

Similar ideas give O(log n) for max flow on unrelated machines

Gaps: Total flow $O(\log P \log n)$ vs $\Omega(\log P)$

Max flow $O(\log n)$ vs $\Omega(1)$

Iterated rounding: simple yet very powerful

Thanks for your attention!