Online and Offline List Batching

Wolfgang Bein

Center for the Advanced Study of Algorithms
School of Computer Science
University of Nevada, Las Vegas

2007

supported by NSF grant CCR-0312093

Osaka Kinko's

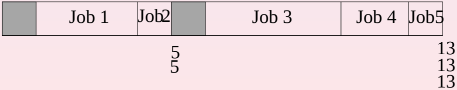
List Batching

n jobs are given to be processed in batches

Job 1 Job 2 Job 3 Job 4 Job 5

all jobs in a batch finish at the same time

there is a setup time to get a batch started



the object is to minimize the average completion time

List Batching, continued...

Jobs with processing requirements $p_1, p_2, \ldots p_n$ are given and have to processed in that order.

List Batching, continued...

Jobs with processing requirements $p_1, p_2, \ldots p_n$ are given and have to processed in that order.

There is one machine.

List Batching, continued...

Jobs with processing requirements $p_1, p_2, \ldots p_n$ are given and have to processed in that order.

There is one machine.

Jobs are given to the machine in batches. Every batch has a setup time of 1.

List Batching, continued...

Jobs with processing requirements $p_1, p_2, \ldots p_n$ are given and have to processed in that order.

There is one machine.

Jobs are given to the machine in batches. Every batch has a setup time of 1.

- The completion time C_i of job i is the completion time of its batch.

List Batching, continued...

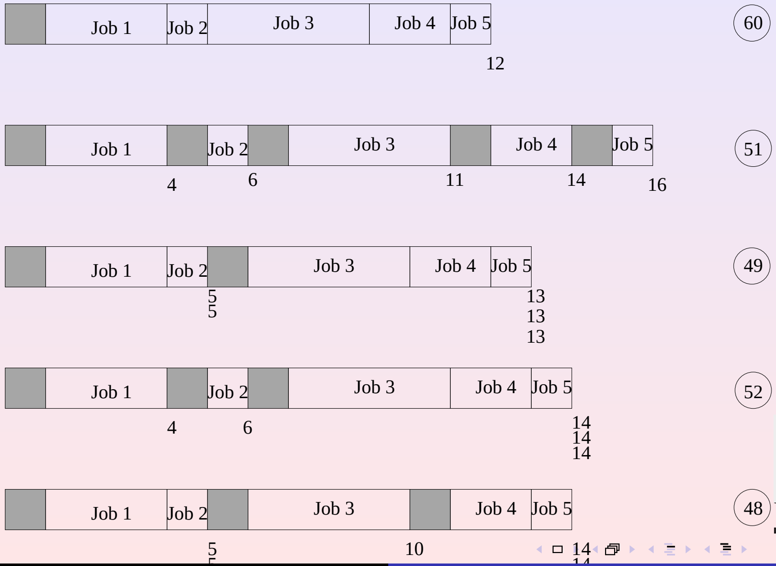
Jobs with processing requirements $p_1, p_2, \ldots p_n$ are given and have to processed in that order.

There is one machine.

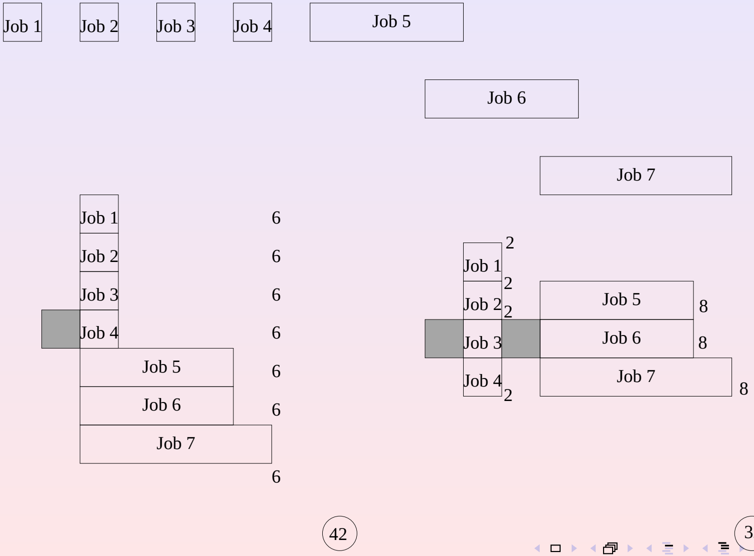
Jobs are given to the machine in batches. Every batch has a setup time of 1.

- The completion time C_i of job i is the completion time of its batch.
- The object is to batch the jobs in such a way that $\sum C_i$ is minimized.

Our Example


Job 1
Job 2
Job 3
Job 4
Job 5
60
12
Job 1
Job 2
Job 3
Job 4
Job 5
51
4
6
11
14
16
Job 1
Job 2
Job 3
Job 4
Job 5
49
5
5
13
13
13
Job 1
Job 2
Job 3
Job 4
Job 5
52
4
6
14
14
14
Job 1
Job 2
Job 3
Job 4
Job 5
48
5
10
14

Sor far List s-Batching, but here is also List p-Batching



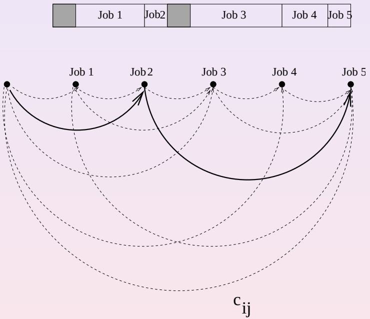
History

Large body of work on offline batching, i.e.

- [Coffman, Yannakakis, Magazine, Santos, 1990]
- [Albers, Brucker, 1993]
- [Brucker, Gladky, Hoogeveen, Kovalyov, Pots, Tautenhahn, Velde, 1998]

List s-Batching, Offline

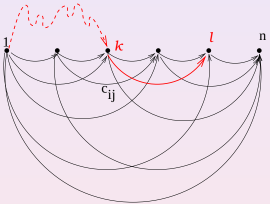
The offline list s-batching problem can be reduced to a path problem[1][AB92]:

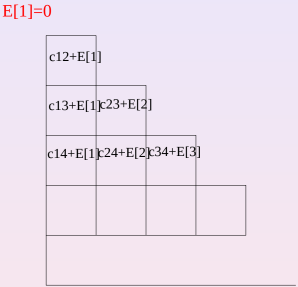


- $c_{ij} = (n - i)(s + P_j - P_i)$ with $P_i = \sum_{\ell=0}^{i} p_\ell$

[1] List p-Batching has a similar reduction

A Simple Dynamic Program

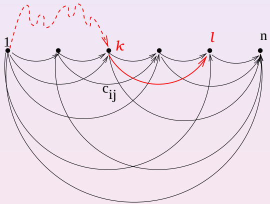


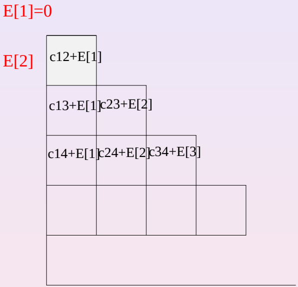


$$E[\ell] = \text{the shortest path from 1 to } \ell$$
$$E[\ell] = \min_{1 \le k < \ell}\{E[k] + c_{k\ell}\}$$

$O(n^2)$

A Simple Dynamic Program

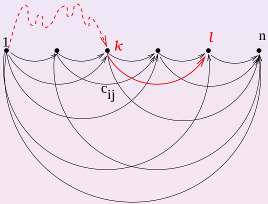




$$E[\ell] = \text{the shortest path from 1 to } \ell$$
$$E[\ell] = \min_{1 \leq k < \ell}\{E[k] + c_{k\ell}\}$$

$O(n^2)$

A Simple Dynamic Program



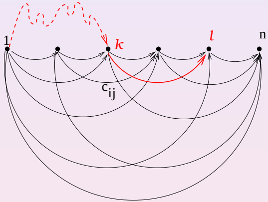
E[1]=0				
E[2]	c12+E[1]			
E[3]	c13+E[1]	c23+E[2]		
	c14+E[1]	c24+E[2]	c34+E[3]	

$$E[\ell] = \text{the shortest path from 1 to } \ell$$

$$E[\ell] = \min_{1 \leq k < \ell}\{E[k] + c_{k\ell}\}$$

$O(n^2)$

A Simple Dynamic Program



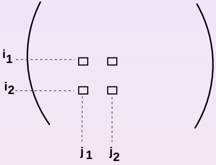
E[1]=0				
E[2]	c12+E[1]			
E[3]	c13+E[1]	c23+E[2]		
E[4]	c14+E[1]	c24+E[2]	c34+E[3]	

$$E[\ell] = \text{the shortest path from 1 to } \ell$$
$$E[\ell] = \min_{1 \leq k < \ell} \{E[k] + c_{k\ell}\}$$

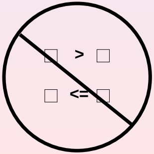
$O(n^2)$

How to do this in $O(n \log n)$

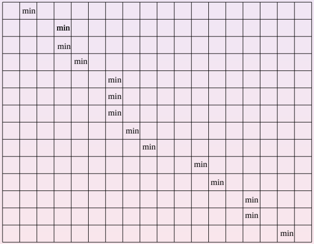
Monge Property



$$c_{i_1 j_1} + c_{i_2 j_2} \le c_{i_2 j_1} + c_{i_1 j_2}$$

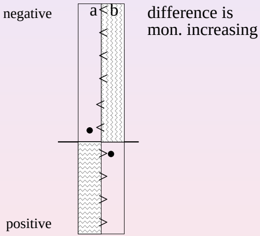


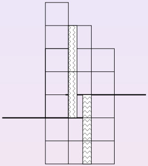
Totally Monotone

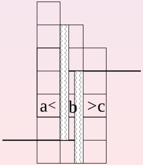


Various Inferences

Entire colums can be eliminated in $O(\log n)$ time:







The Online Protocol of the Dynamic Program

[LS91]

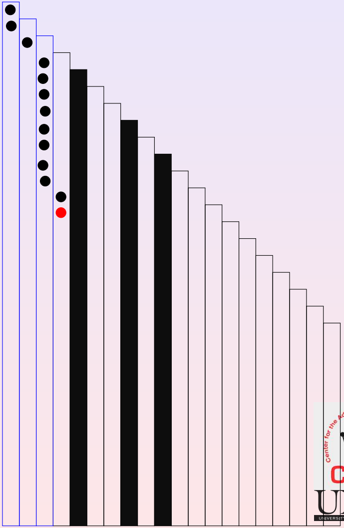
Protocol:
Once the minimum of the

i^{th} row

is known,
the

$(i+1)^{st}$

column is available .



The Online Protocol of the Dynamic Program

[LS91]

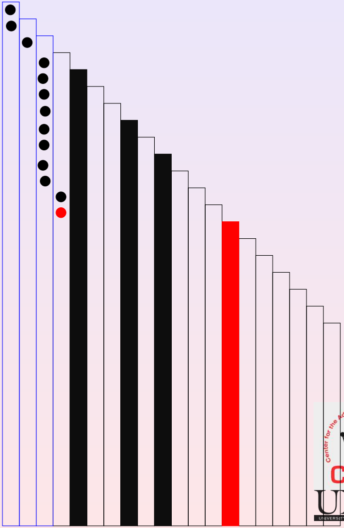
Protocol:
Once the minimum of the

i^{th} row

is known,
the

$(i+1)^{st}$

column is available .



The Online Protocol of the Dynamic Program

[LS91]

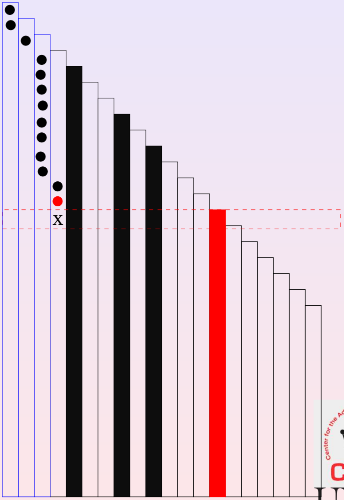
Protocol:
Once the minimum of the

i^{th} row

is known,
the

$(i+1)^{st}$

column is available .



The Online Protocol of the Dynamic Program

[LS91]

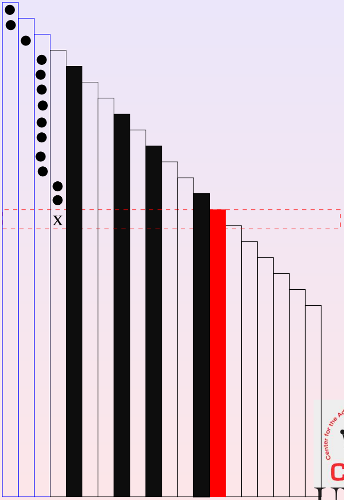
Protocol:
Once the minimum of the

$$i^{th} \text{ row}$$

is known,
the

$$(i+1)^{st}$$

column is available .



The Online Protocol of the Dynamic Program

[LS91]

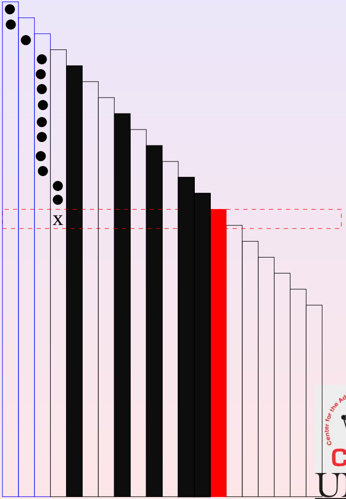
Protocol:
Once the minimum of the

i^{th} row

is known,
the

$(i+1)^{st}$

column is available .



The Online Protocol of the Dynamic Program

[LS91]

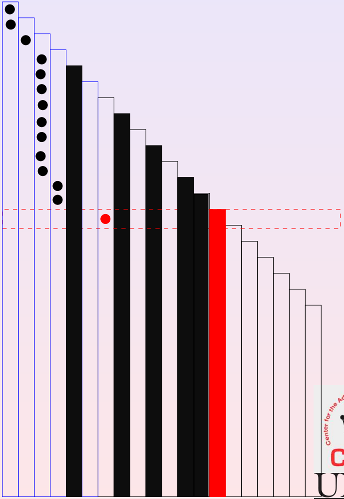
Protocol:
Once the minimum of the

i^{th} row

is known,
the

$(i+1)^{st}$

column is available .



The Online Protocol of the Dynamic Program

[LS91]

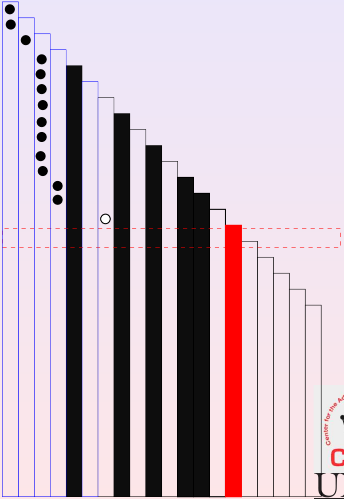
Protocol:
Once the minimum of the

$$i^{th} \text{ row}$$

is known,
the

$$(i+1)^{st}$$

column is available .



Algorithm is $O(n \log n)$

The Hire/Fire/Retire Algorithm can be implemented in $O(n \log n)$

- Potential: number of rows + number of columns.
- Retire eliminates a column, not-retire eliminates a row, fire eliminates a column, not-fire eliminates a row.

$O(n)$ Algorithms:

- [LARSH 91]
- [Albers, Brucker 93]

A closed form for $p_i = s = 1$

Theorem ([BELN 04])

$$optcost[n] = \frac{m(m+1)(m+2)(3m+5)}{24} + k(n+m-k+1) + \frac{k(k+1)}{2}$$

$$\text{for } n = \frac{m(m+1)}{2} + k$$

The optimal size of the first batch

$$= \begin{cases} m \text{ if } k = 0 \\ m \text{ or } m+1 \text{ if } 0 < k < m+1 \\ m+1 \text{ if } k = m+1 \end{cases}$$

Online List Batching

- Jobs $J_1, J_2, \ldots$ arrive one by one over a list.

Online List Batching

- Jobs $J_1, J_2, \ldots$ arrive one by one over a list.
- Job J_i must be scheduled before a new job is seen, and even before knowing whether current is the last job.

Online List Batching

- Jobs $J_1, J_2, \ldots$ arrive one by one over a list.
- Job J_i must be scheduled before a new job is seen, and even before knowing whether current is the last job.
- For job J_i an online Algorithm must decide whether to

  "batch": to make J_i the first job of a new bach
  "not to batch": to add J_i to the current batch.

Competitiveness

- A measure of the performance that compares the decision made online with the optimal offline solution for the same problem.

For any sequence of jobs $\rho = \{J_1, J_2, \ldots\}$

$cost_{\mathcal{A}}(\rho)$: cost of the schedule produced by $\mathcal{A}$ for ρ
$cost_{opt}(\rho)$ is the minimum cost of any schedule for ρ

We say that $\mathcal{A}$ is C-competitive if for each sequence ρ we have

$$cost_{\mathcal{A}}(\rho) \leq C \cdot cost_{opt}(\rho)$$

Algorithm PSEUDOBATCH(B)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

Algorithm PSEUDOBATCH(B)

Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

Algorithm PSEUDOBATCH(B)

Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

Algorithm PSEUDOBATCH(B)

Pseudobatch(1)

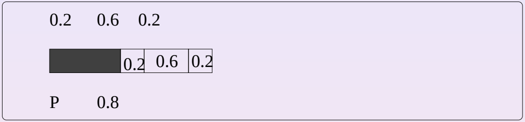
- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

Algorithm PSEUDOBATCH(B)

Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

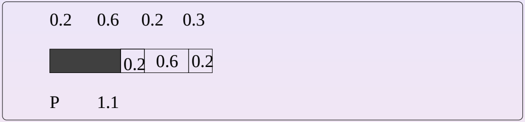
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

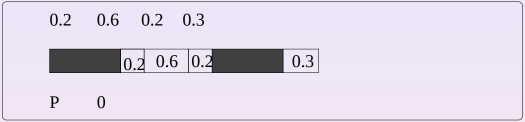
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

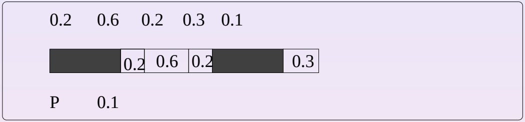
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

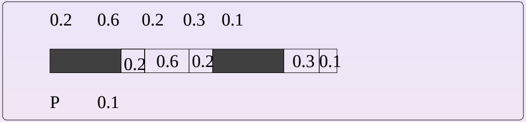
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

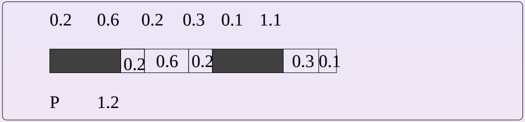
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

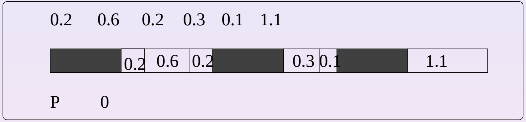
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

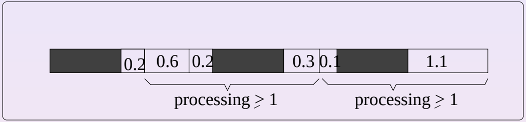
Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

Algorithm PSEUDOBATCH(B)



Pseudobatch(1)

- PSEUDOBATCH(B) maintains a variable P which will be the sum of the processing times of a set of recent jobs.
- When J_1 is received, P is set to 0. After receiving each subsequent J_i, PSEUDOBATCH(B) first adds p_i to P.
- If $P > B$, PSEUDOBATCH(B) batches and also sets P to zero.

PSEUDOBATCH(1) is 2-competitive

Theorem ([BELN 04])

The competitiveness of algorithm PSEUDOBATCH(1) *is not larger than 2*

PSEUDOBATCH(1) is 2-competitive

Theorem ([BELN 04])

The competitiveness of algorithm PSEUDOBATCH(1) *is not larger than 2*

Proof.

Let $S_i = \sum_{j=1}^{i} p_j$.

PSEUDOBATCH(1) is 2-competitive

Theorem ([BELN 04])

The competitiveness of algorithm PSEUDOBATCH(1) *is not larger than 2*

Proof.

Let $S_i = \sum_{j=1}^{i} p_j$.

- Optimal Completion Times: $C_i^* \geq 1 + S_i$

PSEUDOBATCH(1) is 2-competitive

Theorem ([BELN 04])

The competitiveness of algorithm PSEUDOBATCH(1) *is not larger than 2*

Proof.

Let $S_i = \sum_{j=1}^{i} p_j$.

- Optimal Completion Times: $C_i^* \geq 1 + S_i$
- For PSEUDOBATCH(1): $C_i \leq \#\text{batches} + S_i + 1$

□

PSEUDOBATCH(1) is 2-competitive

Theorem ([BELN 04])

The competitiveness of algorithm PSEUDOBATCH(1) *is not larger than 2*

Proof.

Let $S_i = \sum_{j=1}^{i} p_j$.

- Optimal Completion Times: $C_i^* \geq 1 + S_i$
- For PSEUDOBATCH(1): $C_i \leq \#\text{batches} + S_i + 1$
- $\#\text{batches} \leq 1 + S_i$

□

PSEUDOBATCH(1) is 2-competitive

Theorem ([BELN 04])

The competitiveness of algorithm PSEUDOBATCH(1) *is not larger than 2*

Proof.

Let $S_i = \sum_{j=1}^{i} p_j$.

- Optimal Completion Times: $C_i^* \geq 1 + S_i$
- For PSEUDOBATCH(1): $C_i \leq \#\text{batches} + S_i + 1$
- $\#\text{batches} \leq 1 + S_i$
- Thus $C_i \leq 2 + 2S_i$, which implies the result.

□

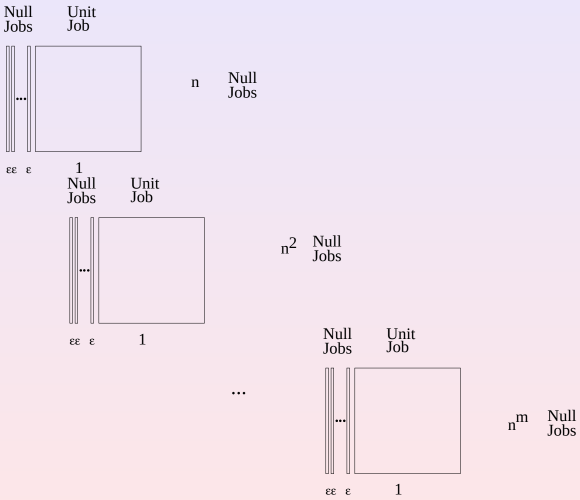
PSEUDOBATCH(1) is Optimal

Theorem ([BELN 04])

The competitiveness of any deterministic online algorithm for the list s-batch problem is at least 2.

- Construct an adversary such that any deterministic algorithm will perform “poorly”.
- Aversary uses Null Jobs.
- Null Jobs are jobs with “arbitrarily” small processing times.

Lower Bound Adversary



Proof Sketch

Proof.

- Let m be a large integer; the sequence ends

  a: the first time $\mathcal{A}$ does not batch,

  b: or at m.

□

Proof Sketch

Proof.

- Let m be a large integer; the sequence ends
  - a: the first time $\mathcal{A}$ does not batch,
  - b: or at m.
- in case a we have $cost_{\mathcal{A}} = n^k(k + k) +$ low order

Proof Sketch

Proof.

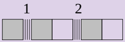
- Let m be a large integer; the sequence ends
  - a: the first time $\mathcal{A}$ does not batch,
  - b: or at m.
- in case a we have $cost_{\mathcal{A}} = n^k(k + k) + \text{low order}$

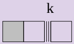
- *opt* places all but the last job into one batch,
  $cost_{\mathcal{A}} = n^k(k) + \text{low order}$

Proof Sketch

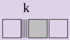
Proof.

- Let m be a large integer; the sequence ends
  - a: the first time $\mathcal{A}$ does not batch,
  - b: or at m.
- in case a we have $cost_{\mathcal{A}} = n^k(k+k) +$ low order





- *opt* places all but the last job into one batch,
  $cost_{\mathcal{A}} = n^k(k) +$ low order



- case b similar...

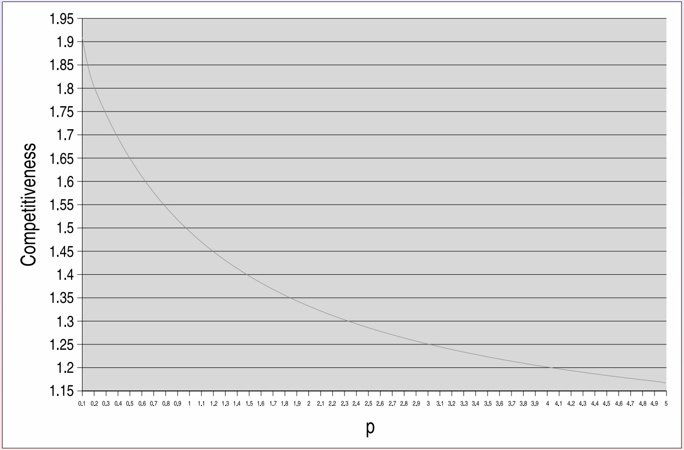
Small jobs are needed...

- The next result shows that the exact competitiveness of 2 relies on the fact that the jobs may be arbitrarily small.
- In fact, if there is a positive lower bound on the size of the jobs, it is possible to construct an algorithm with competitiveness less than two.

Theorem ([BELN 04])

If the processing time of every job is at least p, then $\mathcal{A} =$ PSEUDOBATCH$(\sqrt{p+1})$ *is C-competitive, where*
$$C = \min\left(\frac{1+\sqrt{p+1}}{\sqrt{p+1}}, \frac{p+1}{p}\right).$$

If jobs are at least *p*...



The uniform case of $p_i = s = 1$

Define $\mathcal{D}$ to be the online algorithm which batches after jobs: 2, 5, 9, 13, 18, 23, 29, 35, 41, 48, 54, 61, 68, 76, 84, 91, 100, 108, 117, 126, 135, 145, 156, 167, 179, 192, 206, 221, 238, 257, 278, 302, 329, 361, 397, 439, 488, 545, 612, 690, 781, 888, 1013, 1159, 1329, 1528, 1760, and 2000+40i for all $i \geq 0$.

Algorithm was found by computer

Theorem ([BELN 04])

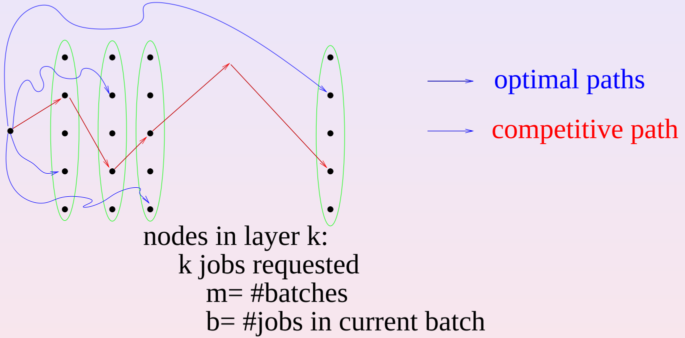
$\mathcal{D}$ is $\frac{619}{583}$-competitive, and no online algorithm the list batching problem restricted to unit job sizes has competitiveness smaller than $\frac{619}{583}$.

The case pf $p_i = s = 1$; upper bound

- It is easy to show that $\frac{619}{583}$ is an uppper bound on the competitive ratio of the algorithms if there are more than 2000 jobs.
- The ratio is only tight when there are fewer than 2000 jobs.
- The algorithm was found by computer simulation.

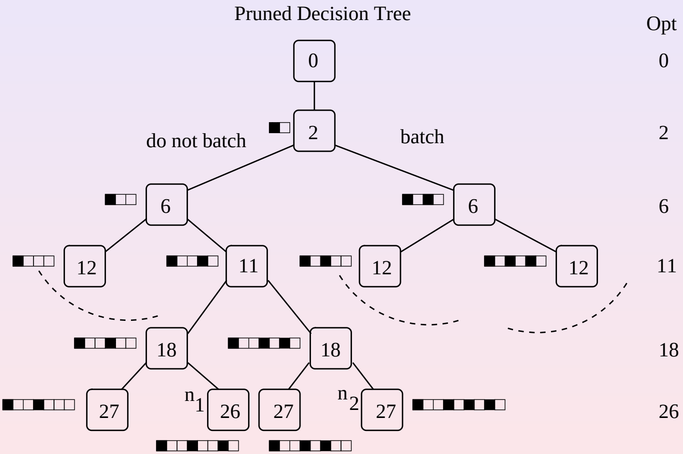
The case pf $p_i = s = 1$; Algorithm

Minimum Competitiveness Layered Graph Problem



- Schedule are combined into classes.
- A class has schedules where there are m batches, the last batch contains b jobs, and k jobs have been requested.

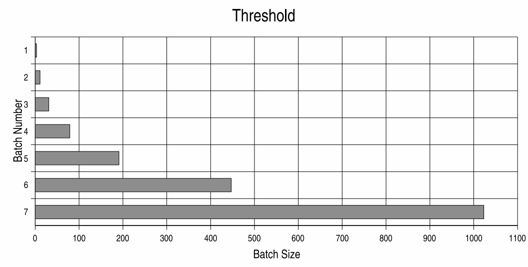
The case pf $p_i = s = 1$; Lower Bound Proof



Competitiveness for p-Batching

THRESHOLD:

batches for the ℓ^{th} time whenever the processing requirement of the next job $\geq (\ell+1)2^{\ell}-1$,
i.e. 3, 11, 31, 79, . . .



Competitiveness for p-Batching

Theorem ([BELN 04])

THRESHOLD *is 4-competitive. No deterministic online algorithm for the list p-batch problem can have competitiveness less than 4.*

Proof.

Lower bound proof a bit subtle.... □

Open Problems: Weighted Batching

Problem

Given n jobs with

1. *processing times* $p_1, \dots p_n$
2. *non-negative weights* $w_1 \dots w_n$.
3. *offline*

Find an order and s-batching that minimizes $\sum w_i C_i$.

- Problem is NP-hard.
- Sort jobs in order of "priorities" $\frac{w_i}{p_i}$ then PSEUDOBATCH(1) is a 2-approximation.

PTAS ?