Online and Offline List Batching

Wolfgang Bein

Department of Computer Science
University of Nevada, Las Vegas
Today min: 26 C max 42 C

Universität Lübeck June 16, 2015

( Today min: 4.9C max 14.9 C, previous talk November 14, 2014: min 7.2 C max 11.6 C)

supported by NSF grant CCR-0312093
List Batching

$n$ jobs are given to be processed in batches

<table>
<thead>
<tr>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Job 4</th>
<th>Job 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
Jobs with processing requirements $p_1, p_2, \ldots, p_n$ are given and have to be 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.
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.
Sor far List s-Batching, but here is also **List p-Batching**

---

**Online and Offline List 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]
The offline list s-batching problem can be reduced to a path problem\cite{ab92}:

\[ c_{ij} = (n - i)(s + P_j - P_i) \text{ with } P_i = \sum_{\ell=0}^{i} p_{\ell} \]
A Simple Dynamic Program

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

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

\[ E[1] = 0 \]

\[ c_{12} + E[1] \]

\[ c_{13} + E[1] + c_{23} + E[2] \]


\[ O(n^2) \]
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

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

\[ E[1] = 0 \]
\[ E[2] \]
\[ c_{12} + E[1] \]
\[ c_{13} + E[1] + c_{23} + E[2] \]

\[ O(n^2) \]
A Simple Dynamic Program

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

\[ O(n^2) \]
How to do this in $O(n \log n)$

Monge Property

$C_{i_1 j_1} + C_{i_2 j_2} \leq C_{i_2 j_1} + C_{i_1 j_2}$

Totally Monotone
Entire columns can be eliminated in $O(\log n)$ time:
[LS91]

 Protocol: Once the minimum of the $i^{th}$ row is known, the $(i + 1)^{st}$ column is available.
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.
Protocol:
Once the minimum of the $i^{th}$ row is known, the $(i + 1)^{st}$ column is available.

[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.
Algorithm is $O(n \log n)$

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

- Potential (for number of hire/fire/retire operations):
  - number of rows + number of columns.
- Retire eliminates a column, fire eliminates a column, not-retire eliminates a row, not-fire happens once per row.

$O(n)$ Algorithms:

- [LARSH 91]
- [Albers, Brucker 93]
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 batch
  - "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 $\rho = \{J_1, J_2, \ldots\}$

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

We say that $A$ is $C$-competitive if for each sequence $\rho$ we have

$$cost_A(\rho) \leq C \cdot cost_{opt}(\rho)$$
Algorithm $\text{PSEUDOBATCH}(B)$

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

$\textsc{Pseudobatch}(1)$

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

- \textbf{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 $J_i$, \textbf{PSEUDOBATCH}(B) first adds $p_i$ to $P$.
- If $P > B$, \textbf{PSEUDOBATCH}(B) batches and also sets $P$ to zero.
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 \textsc{PSEUDOBATCH}(B)

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

\textbf{Pseudobatch(1)}

\begin{center}
\begin{tabular}{cccc}
0.2 & 0.6 & 0.2 & 0.3 \\
\hline
0.2 & 0.6 & 0.2 \\
P & 1.1
\end{tabular}
\end{center}

- \textbf{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 $J_i$, \textbf{Pseudobatch}(B) first adds $p_i$ to $P$.
- If $P > B$, \textbf{Pseudobatch}(B) batches and also sets $P$ to zero.
Algorithm \textsc{Pseudobatch}(B)

\begin{itemize}
  \item \textsc{Pseudobatch}(B) maintains a variable \( P \) which will be the sum of the processing times of a set of recent jobs.
  \item When \( J_1 \) is received, \( P \) is set to 0. After receiving each subsequent \( J_i \), \textsc{Pseudobatch}(B) first adds \( p_i \) to \( P \).
  \item If \( P > B \), \textsc{Pseudobatch}(B) batches and also sets \( P \) to zero.
\end{itemize}
**Algorithm** \( \text{PSEUDOBATCH}(B) \)

\[
\begin{array}{cccc}
0.2 & 0.6 & 0.2 & 0.3 & 0.1 \\
0.2 & 0.6 & 0.2 & 0.3 \\
\end{array}
\]

**Pseudobatch(1)**

- \( \text{PSEUDOBATCH}(B) \) 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 \( J_i \), \( \text{PSEUDOBATCH}(B) \) first adds \( p_i \) to \( P \).
- If \( P > B \), \( \text{PSEUDOBATCH}(B) \) batches and also sets \( P \) to zero.
**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 \textsc{Pseudobatch}(B)

\begin{itemize}
  \item \textsc{Pseudobatch}(B) maintains a variable $P$ which will be the sum of the processing times of a set of recent jobs.
  \item When $J_i$ is received, $P$ is set to 0. After receiving each subsequent $J_i$, \textsc{Pseudobatch}(B) first adds $p_i$ to $P$.
  \item If $P > B$, \textsc{Pseudobatch}(B) batches and also sets $P$ to zero.
\end{itemize}
Algorithm \textsc{Pseudobatch}(B)

\begin{itemize}
  \item \textsc{Pseudobatch}(B) maintains a variable $P$ which will be the sum of the processing times of a set of recent jobs.
  \item When $J_1$ is received, $P$ is set to 0. After receiving each subsequent $J_i$, \textsc{Pseudobatch}(B) first adds $p_i$ to $P$.
  \item If $P > B$, \textsc{Pseudobatch}(B) batches and also sets $P$ to zero.
\end{itemize}
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.
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 $S_i = \sum_{j=1}^{i} p_j$. 

Wolfgang Bein

Online and Offline List Batching
**Theorem ([BELN 04])**

The competitiveness of algorithm \textsc{Pseudobatch}(1) is not larger than 2.

**Proof.**

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

- Optimal Completion Times: $C_i^* \geq 1 + S_i$
Theorem ([BELN 04])

The competitiveness of algorithm \textsc{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 \textsc{Pseudobatch}(1): \( C_i \leq \text{#batches} + S_i + 1 \)
**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): $C_i \leq \#batches + S_i + 1$
- $\#batches \leq 1 + S_i$
**Theorem ([BELN 04])**

The competitiveness of algorithm $\text{PSEUDOBATCH}(1)$ 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 $\text{PSEUDOBATCH}(1)$: $C_i \leq \#\text{batches} + S_i + 1$
- $\#\text{batches} \leq 1 + S_i$
- 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”.
- Adversary uses **Null Jobs**.
- Null Jobs are jobs with “arbitrarily” small processing times.
Lower Bound Adversary

\[ \text{Null Jobs} \quad \text{Unit Job} \]

\[ \varepsilon \quad \varepsilon \quad 1 \quad \varepsilon \quad \varepsilon \]

\[ \varepsilon \quad \varepsilon \quad 1 \quad \varepsilon \quad \varepsilon \]

\[ \varepsilon \quad \varepsilon \quad 1 \quad \varepsilon \quad \varepsilon \]

\[ \varepsilon \quad \varepsilon \quad 1 \quad \varepsilon \quad \varepsilon \]

\[ \text{n Jobs} \quad \text{n Jobs} \]

\[ \text{n^2 Jobs} \quad \text{n^2 Jobs} \]

\[ \text{n^m Jobs} \]
Proof Sketch

Proof.

Let \( m \) be a large integer; the sequence ends

a: the first time \( A \) 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 \( \text{cost}_\mathcal{A} = n^k(k + k) + \text{low order} \)

\[
\begin{array}{cccc}
1 & 2 & \cdots & k \\
\hline
\text{boxed} & \text{boxed} & \text{boxed} & \text{boxed} \\
\hline
\end{array}
\]
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 \( \text{cost}_\mathcal{A} = n^k(k + k) + \text{low order} \)

\[
\begin{array}{ccc}
1 & 2 & k \\
\hline
\text{\#} & \text{\#} & \text{\#} \\
\text{\#} & \text{\#} & \text{\#} \\
\end{array}
\]

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

\[
\begin{array}{ccc}
1 & 2 & k \\
\hline
\text{\#} & \text{\#} & \text{\#} \\
\text{\#} & \text{\#} & \text{\#} \\
\end{array}
\]
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, $cost_\mathcal{A} = n^k(k) + \text{low order}$

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 $A = \text{PSEUDOBATCH}(\sqrt{p + 1})$ 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 $i \geq 0$.

**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 $\frac{619}{583}$. 
Offline: A closed form for $p_i = s = 1$

**Theorem ([BELN 04])**

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

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: The case $p_i = s = 1$

Algorithm was found by computer
Minimum Competitiveness Layered Graph Problem

- Schedules 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.
Weighted Batching

**Problem**

Given $n$ jobs with

1. processing times $p_1, \ldots, p_n$
2. non-negative weights $w_1, \ldots, w_n$.

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

Problem is NP-hard.
Weighted Batching

Priorities \( \frac{w_i}{p_i} \)

- Sort jobs in order of “priorities” \( \frac{w_i}{p_i} \) “Canonical Order”
- Then list batch

If all weights \( w_i \) are equal, then is this optimal.
If all processing times \( p_i \) are equal, then is this optimal.
Done.

- But what if not?
2-Approximations

**CanPseudoBatch**: put the jobs in canonical order and then use PseudoBatch

Theorem

CanPseudoBatch has an approximation ratio of 2.

**CanonicalBest**: put the jobs in canonical order and then use MongeOpt

Theorem

CanonicalBest has an approximation ratio of 2.
A lower bound for Priority Approximation

**Priority Algorithm:** Any algorithm that schedules jobs in canonical order.

**Theorem**

Approximation ratio of any priority algorithm \( \geq \frac{2 + \sqrt{6}}{4} \approx 1.1124 \)

Proof: Two jobs \((p_1, w_1), (p_2, w_2)\) with equal priorities: 
\((1, 1 + \epsilon), (1 + \sqrt{6}, 1 + \sqrt{6})\).

![Diagram showing the approximation ratio proof](attachment://images/diagram.png)
**Threshold:**

batches for the $\ell^{th}$ time whenever the processing requirement of the next job $\geq (\ell + 1)2^\ell - 1$, i.e. 3, 11, 31, 79, ...
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 \( p_1, \ldots, p_n \)
2. non-negative weights \( w_1, \ldots, w_n \).
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 \text{PSEUDOBATCH}(1) is a 2-approximation.

PTAS?