Minimum Energy Scheduling

Marek Chrobak
University of California, Riverside
How to Keep Sheep Warm in a Snow Storm?
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps
How to Keep Sheep Warm in a Snow Storm?

- $n$ sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps
How to Keep Sheep Warm in a Snow Storm?

- n sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps
How to Keep Sheep Warm in a Snow Storm?

- \( n \) sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps
How to Keep Sheep Warm in a Snow Storm?

- $n$ sheep on a line huddle together to stay warm
- Each sheep is chained
- Objective: group sheep to minimize # of gaps
How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can’t work ...

**partial grouping 1**

![Sheep diagram 1]

**partial grouping 2**

![Sheep diagram 2]
How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can’t work ...

partial grouping 1

grouping 1 is better, so far ...

partial grouping 2
How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can't work ...

grouping 1 is better, so far ...
How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can’t work ...

Grouping 1 is better, so far ...
How to Keep Sheep Warm in a Snow Storm?

Intuition -- why naive algorithms can't work ...

partial grouping 1

partial grouping 2

grouping 1 is better, so far ...
but only grouping 2 could be extensible to global optimum

tradeoff: # gaps vs gap sizes
How to Keep Sheep Warm in a Snow Storm?

- \([a_j, b_j]\) = range of sheep \(j\)
- assume \(b_1 \leq b_2 \leq ... \leq b_n\)
- a gap with respect to interval \([u, v]\):
  - internal gap or
  - initial gap or
  - final gap

\[u\] \hspace{1cm} \[3 \text{ gaps w.r.t. } [u, v]\] \hspace{1cm} \[v\]
How to Keep Sheep Warm in a Snow Storm?

\[ \text{Inst}_k(u,v) = \text{all sheep } j \in \{1,2,...,k\} \text{ for which } a_j \in [u,v] \]

\[ \text{Gaps}_k(u,v) = \text{min. number of gaps of } \text{Inst}_k(u,v) \text{ w.r.t } [u,v] \]

\[ \text{Gaps}_F(u,v) = \quad ? \]
How to Keep Sheep Warm in a Snow Storm?

\[ \text{Inst}_k(u,v) = \text{all sheep } j \in \{1,2,...,k\} \text{ for which } a_j \in [u,v] \]

\[ \text{Gaps}_k(u,v) = \text{min. number of gaps of Inst}_k(u,v) \text{ w.r.t } [u,v] \]

\[ \text{Gaps}_F(u,v) = ? \]
How to Keep Sheep Warm in a Snow Storm?

$$\text{Inst}_k(u,v) = \text{all sheep } j \in \{1,2,...,k\} \text{ for which } a_j \in [u,v]$$

$$\text{Gaps}_k(u,v) = \text{min. number of gaps of } \text{Inst}_k(u,v) \text{ w.r.t } [u,v]$$

$$\text{Gaps}_F(u,v) = \ ?$$
How to Keep Sheep Warm in a Snow Storm?

\[ \text{Inst}_k(u,v) = \text{all sheep } j \in \{1,2,...,k\} \text{ for which } a_j \in [u,v] \]

\[ \text{Gaps}_k(u,v) = \text{min. number of gaps of Inst}_k(u,v) \text{ w.r.t } [u,v] \]

\[ \text{Gaps}_F(u,v) = ? \]
How to Keep Sheep Warm in a Snow Storm?

\[ \text{Inst}_k(u,v) = \text{all sheep } j \in \{1, 2, \ldots, k\} \text{ for which } a_j \in [u,v] \]

\[ \text{Gaps}_k(u,v) = \text{min. number of gaps of Inst}_k(u,v) \text{ w.r.t } [u,v] \]

\[ \text{Gaps}_F(u,v) = ? \]
How to Keep Sheep Warm in a Snow Storm?

Inst_k(u,v) = all sheep j ∈ \{1,2,...,k\} for which a_j ∈ [u,v]

Gaps_k(u,v) = min. number of gaps of Inst_k(u,v) w.r.t [u,v]

Gaps_F(u,v) = 1
How to Keep Sheep Warm in a Snow Storm?

Recurrence for $Gaps_k(u,v)$
Recurrence for $\text{Gaps}_k(u,v)$

**Case 1:** $a_k \not\in [u,v]$. Then $k \not\in \text{Inst}_k(u,v)$, so

$$\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)$$
Recurrence for $Gaps_k(u,v)$

**Case 1:** $a_k \notin [u,v]$. Then $k \notin \text{Inst}_k(u,v)$, so $Gaps_k(u,v) = Gaps_{k-1}(u,v)$

**Case 2:** $a_k \in [u,v]$, so $k \in \text{Inst}_k(u,v)$.

If $k$ is scheduled at time $t$ then $\text{Inst}_k(u,v) = \{k\} \cup \text{Inst}_{k-1}(u,t) \cup \text{Inst}_{k-1}(t+1,v)$

How to Keep Sheep Warm in a Snow Storm?
Recurrence for $\text{Gaps}_k(u,v)$

**Case 1:** $a_k \notin [u,v]$. Then $k \notin \text{Inst}_k(u,v)$, so

$$\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)$$

**Case 2:** $a_k \in [u,v]$, so $k \in \text{Inst}_k(u,v)$.

If $k$ is scheduled at time $t$ then

$$\text{Inst}_k(u,v) = \{k\} \cup \text{Inst}_{k-1}(u,t) \cup \text{Inst}_{k-1}(t+1,v)$$

**Philippe's Partition Principle:** Wlog grouping looks like this:

$$\text{Inst}_{k-1}(u,t) = \text{Inst}_{k-1}(u,t-1)$$

(in particular, no $r_i$ at $t$, $i \neq k$)
How to Keep Sheep Warm in a Snow Storm?

Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum $t$ (position of $k$)
Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum $t$ (position of $k$)

If $j \in \text{Inst}_{k-1}(t+1,v)$:
**Case 2**: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum $t$ (position of $k$)

**If** $j \in \text{Inst}_{k-1}(t+1,v)$:

If $j \in \text{Inst}_{k-1}(u,t)$:
**Case 2:** $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum $t$ (position of $k$)

If $j \in \text{Inst}_{k-1}(t+1,v)$:

If $j \in \text{Inst}_{k-1}(u,t)$:
Case 2: \( a_k \in [u,v] \), proof of PPP.

Fix an optimal grouping with maximum \( t \) (position of \( k \))

If \( j \in \text{Inst}_{k-1}(t+1,v) \):

If \( j \in \text{Inst}_{k-1}(u,t-1) \):
Case 2: $a_k \in [u,v]$, proof of PPP.

Fix an optimal grouping with maximum $t$ (position of $k$)

If $j \in \text{Inst}_{k-1}(t+1,v)$:

If $j \in \text{Inst}_{k-1}(u,t-1)$:
How to Keep Sheep Warm in a Snow Storm?

Recurrence for Gaps\(_k(u,v)\)

**Case 1:** \(a_k \notin [u,v]\). Then \(k \notin \text{Inst}_k(u,v)\), so
\[
\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)
\]

**Case 2:** \(a_k \in [u,v]\), so \(k \in \text{Inst}_k(u,v)\).

If \(k\) is scheduled at time \(t\) then
\[
\text{Inst}_k(u,v) = \{k\} \cup \text{Inst}_{k-1}(u,t) \cup \text{Inst}_{k-1}(t+1,v)
\]

**Philippe's Partition Principle:** Wlog grouping looks like this:
How to Keep Sheep Warm in a Snow Storm?

Recurrence for $\text{Gaps}_k(u,v)$

**Case 1:** $a_k \notin [u,v]$. Then $k \notin \text{Inst}_k(u,v)$, so

$$\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)$$

**Case 2:** $a_k \in [u,v]$, so $k \in \text{Inst}_k(u,v)$.

If $k$ is scheduled at time $t$ then

$$\text{Inst}_k(u,v) = \{k\} \cup \text{Inst}_{k-1}(u,t) \cup \text{Inst}_{k-1}(t+1,v)$$

**Philippe’s Partition Principle:** Wlog grouping looks like this:

So $\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,t-1) + \text{Gaps}_{k-1}(t+1,v)$
Recurrence for $\text{Gaps}_k(u,v)$

**Case 1:** $a_k \notin [u,v]$. Then $k \notin \text{Inst}_k(u,v)$, so

$$\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)$$

**Case 2:** $a_k \in [u,v]$, so $k \in \text{Inst}_k(u,v)$.

If $k$ is scheduled at time $t$ then

$$\text{Inst}_k(u,v) = \{k\} \cup \text{Inst}_{k-1}(u,t) \cup \text{Inst}_{k-1}(t+1,v)$$

**Philippe's Partition Principle:** Wlog grouping looks like this:

$$\text{Inst}_{k-1}(u,t) = \text{Inst}_{k-1}(u,t-1)$$

$$\text{Inst}_{k-1}(t+1,v)$$

So $\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,t-1) + \text{Gaps}_{k-1}(t+1,v)$

What's $t$? Try all !!!
How to Keep Sheep Warm in a Snow Storm?

Algorithm B0:

if \( a_k \not\in [u,v] \) then

\[
Gaps_k(u,v) = Gaps_{k-1}(u,v)
\]

if \( a_k \in [u,v] \) then

\[
Gaps_k(u,v) = \min_t \{ Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v) \}
\]

where \( a_k \leq t \leq \min(v,b_t) \)

Output \( Gaps_n( a_{\min-1} , b_{\max+1} ) \)
Algorithm B0:

if \( a_k \not\in [u,v] \) then

\[
Gaps_k(u,v) = Gaps_{k-1}(u,v)
\]

if \( a_k \in [u,v] \) then

\[
Gaps_k(u,v) = \min_t \{ Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v) \}
\]

where \( a_k \leq t \leq \min(v,b_t) \)

Output \( Gaps_n( a_{\min-1} , b_{\max+1} ) \)

Time \( O( (n \cdot k') \cdot (R \cdot u') \cdot (R \cdot v') \cdot (R \cdot t') ) = O(nR^3) \) for \( R = b_{\max} - a_{\min} \)
How to Keep Sheep Warm in a Snow Storm?

**Algorithm B0:**

if $a_k \not\in [u,v]$ then

$$Gaps_k(u,v) = Gaps_{k-1}(u,v)$$

if $a_k \in [u,v]$ then

$$Gaps_k(u,v) = \min_t \{ Gaps_{k-1}(u,t-1) + Gaps_{k-1}(t+1,v) \}$$

where $a_k \leq t \leq \min(v,b_t)$

Output $Gaps_n( a_{\min-1} , b_{\max+1} )$

Time $O( (n k's) \cdot (R u's) \cdot (R v's) \cdot (R t's) ) = O(nR^3)$ for $R = b_{\max} - a_{\min}$

Call set $A$ a **minimizer set** if choosing $u,v,t$ from $A$ does not increase the solution

Can we find a smaller minimizer set?
How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets
How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets
shift each group right:
How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets
shift each group right:

\[ b_{A \pm z}, z \leq n \]
How to Keep Sheep Warm in a Snow Storm?

Reducing minimizer sets
shift each group right:

So \( \{b_j \pm z's\} = \{b_j \pm z : j, z \leq n\} \) is a minimizer set for \( t \)

\[ |\{b_j \pm z's\}| = O(n^2) \]
Algorithm B1:

if $a_k \not\in [u,v]$ then

$\text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v)$

if $a_k \in [u,v]$ then

$\text{Gaps}_k(u,v) = \min_t \{ \text{Gaps}_{k-1}(u,t-1) + \text{Gaps}_{k-1}(t+1,v) \}$

where $a_k \leq t \leq \min(v,b_t)$

Output $\text{Gaps}_n( a_{\min-1}, b_{\max+1} )$

Choose $u,v,t$ from $\{b_j \pm z's\}$

So running time $= O(n \cdot n^2 \cdot n^2 \cdot n^2) = O(n^7)$ [Baptiste, SODA'06]
Minimum Energy Scheduling
- Overview
Minimum Energy Scheduling

**Instance:** collection of n jobs
- job j has release time $r_j$, deadline $d_j$, processing time $p_j$
- 1 unit of processing costs 1 unit of energy
- turning the system on costs L units of energy

Schedule = when each job is processed and when the processor is on

**Objective:** Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)
Minimum Energy Scheduling

**Instance:** collection of $n$ jobs
- job $j$ has release time $r_j$, deadline $d_j$, processing time $p_j$
- 1 unit of processing costs 1 unit of energy
- turning the system on costs $L$ units of energy

Schedule $=$ when each job is processed and when the processor is on

**Objective:** Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

Structure of an optimal schedule:
Minimum Energy Scheduling

**Instance:** collection of $n$ jobs

- job $j$ has release time $r_j$, deadline $d_j$, processing time $p_j$
- 1 unit of processing costs 1 unit of energy
- turning the system on costs $L$ units of energy

**Schedule =** when each job is processed and when the processor is on

**Objective:** Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

**Structure of an optimal schedule:**

- $\leq L$
- $> L$
- $\leq L$
- idle
- busy
- long
- short
Minimum Energy Scheduling

**Instance:** collection of n jobs
- job j has release time \( r_j \), deadline \( d_j \), processing time \( p_j \)
- 1 unit of processing costs 1 unit of energy
- turning the system on costs \( L \) units of energy

Schedule = when each job is processed and when the processor is on

**Objective:** Compute a preemptive schedule that minimizes the total energy usage (assume instance is feasible)

Structure of an optimal schedule:

objective function (ignore green time):

\[
E = \sum_{gaps} g \min(\text{length}(g), L)
\]
Minimum Energy Scheduling

What sheep have to do with it?

For $L = 1$ and all $p_j = 1$:

- $E =$ # of gaps
- $j =$ 
- $r_j = a_j$ and $d_j = b_j$
Minimum Energy Scheduling

What sheep have to do with it?

For $L = 1$ and all $p_j = 1$:

- $E = \# \text{ of gaps}$
- $r_j = a_j$ and $d_j = b_j$

So the case (unit jobs, $L \leq 1$) can be solved in time $O(n^7)$
Minimum Energy Scheduling

What’s known?

Posed as open: Sviridenko [05], Irani, Pruhs [05]

<table>
<thead>
<tr>
<th># proc.</th>
<th>L</th>
<th>( p_j )</th>
<th>assumption</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>any</td>
<td>1</td>
<td></td>
<td>( O(n^7) ) [B’06]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( O(n^4) ) [BCD’08]</td>
</tr>
<tr>
<td>1</td>
<td>any</td>
<td>any</td>
<td></td>
<td>( O(n^5) ) [BCD’08]</td>
</tr>
<tr>
<td>m</td>
<td>any</td>
<td>1</td>
<td></td>
<td>( O(n^7m^5) ) [DG...’07]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>any</td>
<td>agreeable</td>
<td>( O(n\log n) ) [GJS’10]</td>
</tr>
<tr>
<td>1</td>
<td>any</td>
<td>1</td>
<td>agreeable</td>
<td>( O(n^3) ) [GJS’10]</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>1</td>
<td>agreeable</td>
<td>( O(n^3m^2) ) [GJS’10]</td>
</tr>
</tbody>
</table>

[B’06] = Baptiste  
[BCD’08] = Baptiste, Chrobak, Dürr  
[DG...’07] = Demaine, Ghodsi, Hajiaghayi, Sayedi-Roshkhar, Zadimoghaddam  
[GJS’10] = Gururaj, Jalan, Stein
Minimum Energy Scheduling

What’s known?

Posed as open: Sviridenko [05], Irani, Pruhs [05]

<table>
<thead>
<tr>
<th># proc.</th>
<th>L</th>
<th>p_j</th>
<th>assumption</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>any</td>
<td>1</td>
<td></td>
<td>(O(n^7)) [B’06]</td>
</tr>
<tr>
<td></td>
<td>any</td>
<td>any</td>
<td></td>
<td>(O(n^4)) [BCD’08]</td>
</tr>
<tr>
<td>m</td>
<td>any</td>
<td>1</td>
<td></td>
<td>(O(n^7m^5)) [DG...’07]</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>any</td>
<td>agreeable</td>
<td>(O(n\log n)) [GJS’10]</td>
</tr>
<tr>
<td>1</td>
<td>any</td>
<td>1</td>
<td>agreeable</td>
<td>(O(n^3)) [GJS’10]</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>1</td>
<td>agreeable</td>
<td>(O(n^3m^2)) [GJS’10]</td>
</tr>
</tbody>
</table>

\[B’06\] = Baptiste
\[BCD’08\] = Baptiste, Chrobak, Dürr
\[DG...’07\] = Demaine, Ghodsi, Hajiaghayi, Sayedi-Roshkhar, Zadimoghaddam
\[GJS’10\] = Gururaj, Jalan, Stein

\(r_i < r_j \iff d_j < d_j\)
Minimum Energy Scheduling - Techniques
Minimum Energy Scheduling

Main techniques:

* Philippe’s partitioning trick
* Reducing the minimizer sets
* Inversion trick (”large” parameter ⇔ ”small” value)
* $O(n^2)$-time reduction: Energy $\ll$ Gaps
Reducing Minimizer Sets (L=1, p_j=1)

Algorithm B1-L1P1:

if $r_k \not\in [u, v-1]$ then

\[ \text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v) \]

if $r_k \in [u, v-1]$ then

\[ \text{Gaps}_k(u,v) = \min_t \{ \text{Gaps}_{k-1}(u,t) + \text{Gaps}_{k-1}(t+1,v) \} \]

where $r_k \leq t < \min(v, d_t)$

Output $\text{Gaps}_n( r_{min-1} , d_{max+1} )$

Reminder: $u, v, t \in \{d_j\pm z\}$, $|\{d_j\pm z\}| = O(n^2)$
Reducing Minimizer Sets \((L=1, p_j=1)\)

Three WLOG observations:
Reducing Minimizer Sets ($L=1, p_j=1$)

Three WLOG observations:

- $t$ is maximum possible
Reducing Minimizer Sets \((L=1, p_j=1)\)

Three WLOG observations:

- \(t\) is maximum possible
- all deadlines are different
Reducing Minimizer Sets (L=1, p_j=1)

Three WLOG observations:

- t is maximum possible
- all deadlines are different
Reducing Minimizer Sets ($L=1$, $p_j=1$)

Three WLOG observations:

- $t$ is maximum possible
- all deadlines are different
- release times are different
Reducing Minimizer Sets (L=1, p_j=1)

Three WLOG observations:

- t is maximum possible
- all deadlines are different
- release times are different
- v ≤ d_{k+1}
Reducing Minimizer Sets ($L=1$, $p_{j}=1$)

Assume $k \in \text{Inst}_k(u,v)$ and $k$ scheduled at $t$ (latest possible)
Reducing Minimizer Sets (L=1, p_j=1)

Assume $k \in \text{Inst}_k(u,v)$ and $k$ scheduled at $t$ (latest possible).

If $k$ scheduled last then $\text{Inst}_{k-1}(t+1,v)$ is empty.
So assume $k$ is not last.
Reducing Minimizer Sets (L=1, p_j=1)

Assume $k \in \text{Inst}_k(u,v)$ and $k$ scheduled at $t$ (latest possible)

If $k$ scheduled last then $\text{Inst}_{k-1}(t+1,v)$ is empty
So assume $k$ is not last

**Claim:** There is $j$ right after $k$
Reducing Minimizer Sets (L=1, p_j=1)

Assume \( k \in \text{Inst}_k(u,v) \) and \( k \) scheduled at \( t \) (latest possible)

If \( k \) scheduled last then \( \text{Inst}_{k-1}(t+1,v) \) is empty
So assume \( k \) is not last

**Claim:** There is \( j \) right after \( k \)

**Proof:** If not
Reducing Minimizer Sets ($L=1$, $p_j=1$)

Assume $k \in \text{Inst}_k(u,v)$ and $k$ scheduled at $t$ (latest possible)

If $k$ scheduled last then $\text{Inst}_{k-1}(t+1,v)$ is empty
So assume $k$ is not last

**Claim:** There is $j$ right after $k$

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>k</td>
<td>j</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Proof:** If not

```
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>k</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

move $k$ to later (possible, because $d_k$ is largest)
    -- contradiction
Reducing Minimizer Sets (L=1, p_j=1)

Assume \( k \in \text{Inst}_k(u,v) \) and \( k \) scheduled at \( t \) (latest possible)

If \( k \) scheduled last then \( \text{Inst}_{k-1}(t+1,v) \) is empty
So assume \( k \) is not last

**Claim:** There is \( j \) right after \( k \\

From claim: \( t = r_{j-1} \)
so

- minimizer set for \( t \)'s = \( \{r_{j-1}'s\} \)
- minimizer set for \( u \)'s = \( \{r_j's\} \)
Reducing Minimizer Sets (L=1, pj=1)

Algorithm B2-L1P1:

if \( r_k \not\in [u,v-1] \) then

\[ \text{Gaps}_k(u,v) = \text{Gaps}_{k-1}(u,v) \]

if \( r_k \in [u,v-1] \) then

\[ \text{Gaps}_k(u,v) = \min \left\{ \text{Gaps}_{k-1}(u,v-1), \min_t \{ \text{Gaps}_{k-1}(u,t) + \text{Gaps}_{k-1}(t+1,v) \} \right\} \]

where \( a_k \leq t < \min(v,d_t) \)

Output \( \text{Gaps}_n( r_{\min-1} , d_{\max+1} ) \)

Above, choose: \( u \in \{r_j's\} \), \( t \in \{r_j+1's\} \) and \( v \in \{d_j\pm z's\} \)

\[ \Rightarrow \text{Running time} = O( (n k's) \cdot (n u's) \cdot (n^2 v's) \cdot (n t's) ) = O(n^5) \]
Minimum Energy Scheduling

Main techniques:

* Philippe's partitioning trick ✓

* Reducing the minimizer sets ✓

* Inversion trick ("large" parameter ⇔ "small" value)

* $O(n^2)$-time reduction: Energy $\ll$ Gaps
Consider a function $F(a,\ldots) = \min\{f : \text{Yaddi-Yadda}(a,f,\ldots)\}$ s.t.

- $F(a,\ldots)$ is monotone w.r.t. $a$
- range of $a$ is large (exponential)
- range of $F(a,\ldots)$ is small (polynomial)
Inversion Trick

Consider a function $F(a,...) = \min\{f : \text{Yaddi-Yadda}(a,f,...)\}$ s.t.

- $F(a,...)$ is monotone w.r.t. $a$
- Range of $a$ is large (exponential)
- Range of $F(a,...)$ is small (polynomial)

Instead compute $A(f,...) = \min\{a : \text{Yaddi-Yadda}(a,f,...)\}$

and then $F(a,...) = \min\{f : A(f,...) \geq a\}$
(binary search ....)
Inversion Trick (any $p_j$, gaps)

Example 1: Extending Algorithm B2L1P1 (minimizing # gaps, unit jobs) to arbitrary processing times
Inversion Trick (any $p_j$, gaps)

**Example 1:** Extending Algorithm B2L1P1 (minimizing # gaps, unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs

$p_j = 4$
Inversion Trick (any \( p_j \), gaps)

**Example 1:** Extending Algorithm B2L1P1 (minimizing \# gaps, unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs

\[ p_j = 4 \]

This leads to:

- \( \text{Inst}_{k,p}(u,v) = \text{jobs } 1,2,...,k \text{ with } r_j \in [u,v-1], \text{ and with } p_k \leftarrow p \)
- \( G_{k,p}(u,v) = \text{minimum } \# \text{ gaps w.r.t. } [u,v] \text{ for } \text{Inst}_{k,p}(u,v) \)
Inversion Trick (any $p_j$, gaps)

**Example 1:** Extending Algorithm B2L1P1 (minimizing # gaps, unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs

$$p_j = 4$$

This leads to:

- $\text{Inst}_{k,p}(u,v) = \text{jobs } 1,2,\ldots,k \text{ with } r_j \in [u,v-1], \text{ and with } p_k \leftarrow p$
- $G_{k,p}(u,v) = \text{minimum # gaps w.r.t. } [u,v] \text{ for } \text{Inst}_{k,p}(u,v)$

We can apply Algorithm 2 but ... range of $p$ not polynomial
Inversion Trick (any $p_j$, gaps)

**Example 1:** Extending Algorithm B2L1P1 (minimizing # gaps, unit jobs) to arbitrary processing times

Obvious approach: break each job into unit jobs

$p_j = 4$

This leads to:

- $\text{Inst}_{k,p}(u,v) =$ jobs $1, 2, \ldots, k$ with $r_j \in [u, v-1]$, and with $p_k \leftarrow p$
- $G_{k,p}(u,v) =$ minimum # gaps w.r.t. $[u,v]$ for $\text{Inst}_{k,p}(u,v)$

So we invert:

$$A_{k,g}(u,v) = \text{minimum amount } p \text{ of job } k \text{ for which } \text{Inst}_{k,p}(u,v) \text{ has a schedule with } \leq g \text{ gaps}$$
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount $p$ of job k for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) = \text{jobs } 1,2,...,k \text{ with } r_j \in [u,v-1], \text{ and with } p_k \leftarrow p$
- $A_{k,g}(u,v) = \text{minimum amount } p \text{ of job } k \text{ for which }\text{Inst}_{k,p}(u,v) \text{ has a schedule with } \leq g \text{ gaps}$

Assume not whole $k$ executed at the end:
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) =$ jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v) =$ minimum amount $p$ of job $k$ for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole $k$ executed at the end:
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) =$ jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v) =$ minimum amount $p$ of job k for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole k executed at the end:

busy before and after $t$
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) =$ jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v) =$ minimum amount $p$ of job $k$ for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole $k$ executed at the end:

we must have $t = r_1$

busy before and after $t$
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) =$ jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v) =$ minimum amount $p$ of job $k$ for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole $k$ executed at the end:

\[ f \text{ gaps} \quad \rightarrow \quad h \text{ gaps where } f+h = g \]
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v)$ = jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount $p$ of job k for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole k executed at the end:

- $f$ gaps
- $h$ gaps where $f+h = g$
- $A_{k,h}(t,v)$ units of k
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) = \text{jobs } 1,2,...,k \text{ with } r_j \in [u,v-1], \text{ and with } p_k \leftarrow p$
- $A_{k,g}(u,v) =$ minimum amount $p$ of job $k$ for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole $k$ executed at the end:

- $f$ gaps
- $h$ gaps where $f+h = g$

$A_{k,h}(t,v)$ units of $k$

minimum amount of $k$ with $f$ gaps before $t$
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v) =$ jobs 1,2,...,k with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v) =$ minimum amount $p$ of job $k$ for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole $k$ executed at the end:

- $f$ gaps
- $h$ gaps where $f+h = g$

max. compl. time of $\text{Inst}_{k-1}(u,t)$ with $f$ gaps = $C_{k-1,f}(u,t)$

$A_{k,h}(t,v)$ units of $k$
Inversion Trick (any $p_j$, gaps)

- $\text{Inst}_{k,p}(u,v)$ = jobs 1,2,...,$k$ with $r_j \in [u,v-1]$, and with $p_k \leftarrow p$
- $A_{k,g}(u,v)$ = minimum amount $p$ of job $k$ for which $\text{Inst}_{k,p}(u,v)$ has a schedule with $\leq g$ gaps

Assume not whole $k$ executed at the end:

$$f \text{ gaps} \quad \quad \text{h gaps where } f+h = g$$

$$\begin{align*}
\text{Inst}_{k,p}(u,v) &= \text{jobs } 1,2,...,k \text{ with } r_j \in [u,v-1], \text{ and with } p_k \leftarrow p \\
A_{k,g}(u,v) &= \min_t \min_{f+h=g} \big( t - C_{k-1,f}(u,t) + A_{k,h}(t,v) \big) \\
\text{where } t \in \{r_j\}, \ u \in \{r_j\}, \ v \in \{d_j \pm z\}
\end{align*}$$

Also, we need recurrence for $C_{k,f}(u,v)$ using $A(...)$

Running time $O(n^7)$
Inversion Trick (unit jobs, gaps, but faster)

Example 2: Speeding up the unit/gaps case to $O(n^4)$

$$G_k(u,v) = \min_t \{ \# \text{ gaps : yaddi yadda } \}$$
Inversion Trick (unit jobs, gaps, but faster)

Example 2: Speeding up the unit/gaps case to $O(n^4)$

$$G_k(u,v) = \min_t \{ \# \text{ gaps : yaddi yadda } \}$$

Invert: compute

$$V_k(u,g) = \max \{ v : \text{Inst}_k(u,v) \text{ has schedule with } g \text{ gaps } \}$$
Inversion Trick (unit jobs, gaps, but faster)

Example 2: Speeding up the unit/gaps case to $O(n^4)$

$$G_k(u,v) = \min_t \{ \# \text{ gaps} : \text{yaddi yadda} \}$$

$$O(n) \quad O(n) \quad O(n^2) \quad O(n) \quad O(n) \text{ values}$$

Invert: compute

$$V_k(u,g) = \max \{ v : \text{Inst}_k(u,v) \text{ has schedule with } g \text{ gaps} \}$$

Gives $O(n^4)$ [BCD’08]

Can be extended to any $p_j$’s in time $O(n^5)$ [BCD’08]
Minimum Energy Scheduling

Main techniques:

* Philippe’s partitioning trick ✓

* Reducing the minimizer sets ✓

* Inversion trick (“large” parameter ⇔ “small” value) ✓

* $O(n^2)$-time reduction: Energy ≤ Gaps
Reduction: Energy $\preccurlyeq$ Gaps

$S = \text{lex-minimal energy-optimal schedule}$

$[u,v) = \text{short gap in } S$

Claim: wlog, if $r_j < v$ then $j$ is executed before $v$
Reduction: Energy $\preceq$ Gaps

$S = \text{lex-minimal energy-optimal schedule}$

$[u,v) = \text{short gap in } S$

**Claim:** wlog, if $r_j < v$ then $j$ is executed before $v$

**Proof:** suppose not
Reduction: Energy ≤ Gaps

\[ S = \text{lex-minimal energy-optimal schedule} \]
\[ [u,v) = \text{short gap in } S \]

Claim: wlog, if \( r_j < v \) then \( j \) is executed before \( v \)

Proof: suppose not
Reduction: Energy \leq Gaps

S = lex-minimal energy-optimal schedule

\([u,v) = \text{short gap in } S\)

Claim: wlog, if \(r_j < v\) then \(j\) is executed before \(v\)

Proof: suppose not

Then \(E(\text{new } S) \leq E(S)\) and new \(S\) is lex-smaller than \(S\)
--- contradiction
Reduction: Energy ≲ Gaps

\[ S = \text{lex-minimal energy optimal schedule} \]
\[ [u,v) = \text{short gap in } S \]

So \( S \) looks like this

- Jobs released before \( v \)
- Jobs released at or after \( v \)

\( U \leq L \)

No release times
Reduction: Energy $\leq$ Gaps

$S = \text{lex-minimal energy-optimal schedule}$

So $S$ looks like this:

Schedule with $G(u,v)$ gaps and maximum completion time $C(u,v)$
Reduction: Energy $\leq$ Gaps

$S = \text{lex-minimal energy-optimal schedule}$

So $S$ looks like this:

\[ u = r_s, \quad v = r_j \]

schedule with $G(u,v)$ gaps and maximum completion time $C(u,v)$

Denote $E_s = \text{minimum energy schedule of jobs released } \geq r_s$
Reduction: Energy \leq Gaps

S = lex-minimal energy-optimal schedule
So S looks like this:

\[ S > L > L > L \]

\[ \text{Inst}(u,v) \]

\[ u = r_s \]

\[ v = r_j \]

schedule with \( G(u,v) \) gaps and maximum completion time \( C(u,v) \)

Denote \( E_s = \) minimum energy schedule of jobs released \( \geq r_s \)

\[
E_s = \min_{r_j > r_s} \{ L \cdot [ G(r_s, r_j) - 1 ] + [ r_j - C(r_s, r_j) ] + E_j \} 
\]
Reduction: Energy \leq Gaps

S = lex-minimal energy-optimal schedule
So S looks like this:

\[
\begin{align*}
S &= r_s > L > L > L > L\\
\text{Inst}(u,v) &= u = r_s, v = r_j
\end{align*}
\]

schedule with \( G(u,v) \) gaps and
maximum completion time \( C(u,v) \)

Denote \( E_s \) = minimum energy schedule of jobs released \( \geq r_s \)

\[
E_s = \min \{ L \cdot [ G(r_s,r_j) - 1 ] + [ r_j - C(r_s,r_j) ] + E_j \}
\]

\( r_j > r_s \)

Running time: \( O(n^2) + \) (time to compute all \( G() \), \( C() \) values)
Minimum Energy Scheduling

Main techniques:

* Philippe’s partitioning trick  √
* Reducing the minimizer sets  √
* Inversion trick (“large” parameter ⇔ “small” value)  √
* \(O(n^2)\)-time reduction: Energy \(\ll\) Gaps  √
Minimum Energy Scheduling
- Other Results
Claim: WLOG, optimal schedule is compact:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
m processors, unit jobs, gaps [DG...’07]

Claim: WLOG, optimal schedule is compact:

Proof: Suppose not:
Claim: WLOG, optimal schedule is **compact**:

Proof: Suppose not:
Claim: WLOG, optimal schedule is compact:

Proof: Suppose not:
m processors, unit jobs, gaps [DG...’07]

**Claim:** WLOG, optimal schedule is **compact**:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Proof:** Suppose not:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Claim: WLOG, optimal schedule is **compact**:

Proof: Suppose not:

**switch cannot increase # gaps, so repeat till schedule is compact**
Generalize Philippe's partition trick: Sub-instance

m processors, unit jobs, gaps [DG...'07]
Generalize Philippe's partition trick: Sub-instance

m processors, unit jobs, gaps [DG...’07]
Generalize Philippe’s partition trick: Sub-instance

Recurrence:

\[ G_k(u, a, b, v, c) = \min_t \min_{e,f} \{ G( \ldots ) + G( \ldots ) \} \]

Running time \( O(n^7m^5) \) [DG...’07]
m processors, unit jobs, gaps [DG...’07]

Generalize Philippe’s partition trick: Sub-instance

Recurrence:
\[ G_k(u,a,b,v,c) = \min_t \min_{e,f} \left\{ G(\ldots) + G(\ldots) \right\} \]

Running time \( O(n^7m^5) \) [DG...’07]

Can be improved to \( O(n^5m^5) \) using smaller maximizer sets
1 processor, agreeable [GJS’10]

ordered by $r_j$ or $d_j$
Claim 1: Wlog, jobs execute in order 1, 2, 3, ...
Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

Claim 2: Wlog, \( d_j + p_{j+1} \leq d_{j+1} \)
1 processor, agreeable [GJS’10]

Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

Claim 2: Wlog, \( d_j + p_{j+1} \leq d_{j+1} \)
1 processor, agreeable [GJS’10]

Claim 1: Wlog, jobs execute in order 1, 2, 3, ...

Claim 2: Wlog, $d_j + p_{j+1} \leq d_{j+1}$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1 - p_1$
for any other $j$
    if possible, schedule $j$ right after $j-1$
else schedule $j$ at $d_j - p_j$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1 - p_1$
for any other $j$
  if possible, schedule $j$ right after $j-1$
  else schedule $j$ at $d_j - p_j$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1-p_1$
for any other $j$
  if possible, schedule $j$ right after $j-1$
  else schedule $j$ at $d_j-p_j$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1-p_1$
for any other $j$
    if possible, schedule $j$ right after $j-1$
else schedule $j$ at $d_j-p_j$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1-p_1$
for any other $j$
  if possible, schedule $j$ right after $j-1$
  else schedule $j$ at $d_j-p_j$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1 - p_1$
for any other $j$
  if possible, schedule $j$ right after $j-1$
  else schedule $j$ at $d_j - p_j$
Algorithm GJS-L1SO:
- preprocess jobs as in Claim 2
- Schedule 1 at $d_1 - p_1$
- for any other $j$
  - if possible, schedule $j$ right after $j-1$
  - else schedule $j$ at $d_j - p_j$
Algorithm GJS-L1SO:
preprocess jobs as in Claim 2
Schedule 1 at $d_1 - p_1$
for any other $j$
  if possible, schedule $j$ right after $j-1$
  else schedule $j$ at $d_j - p_j$

Running time: sorting + $O(n) = O(n \log n)$
Open (Easy?) Questions
Open (Easy?) Questions

1. Back to sheep: if any group of \( \leq g \) sheep dies, minimize \# of dead sheep
Open (Easy?) Questions

1. Back to sheep: if any group of \( \leq g \) sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
Open (Easy?) Questions

1. Back to sheep: if any group of \( \leq g \) sheep dies, minimize \# of dead sheep
2. Or maximize minimum group size
3. Several power levels
Open (Easy?) Questions

1. Back to sheep: if any group of \( \leq g \) sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
5. Faster algorithms? Can the case (unit jobs, gaps) be solved in time $O(n^3)$?
Open (Easy?) Questions

1. Back to sheep: if any group of $\leq g$ sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
5. Faster algorithms? Can the case (unit jobs, gaps) be solved in time $O(n^3)$?
6. Fast approximations: $1+\varepsilon$-approx. in $\tilde{O}(n)$ time?
Open (Easy?) Questions

1. Back to sheep: if any group of \( \leq g \) sheep dies, minimize # of dead sheep
2. Or maximize minimum group size
3. Several power levels
4. For multiprocessors: each processor can be turned off, or the whole system
5. Faster algorithms? Can the case (unit jobs, gaps) be solved in time \( O(n^3) \)?
6. Fast approximations: 1+\( \varepsilon \)-approx. in \( \tilde{O}(n) \) time?
7. ...