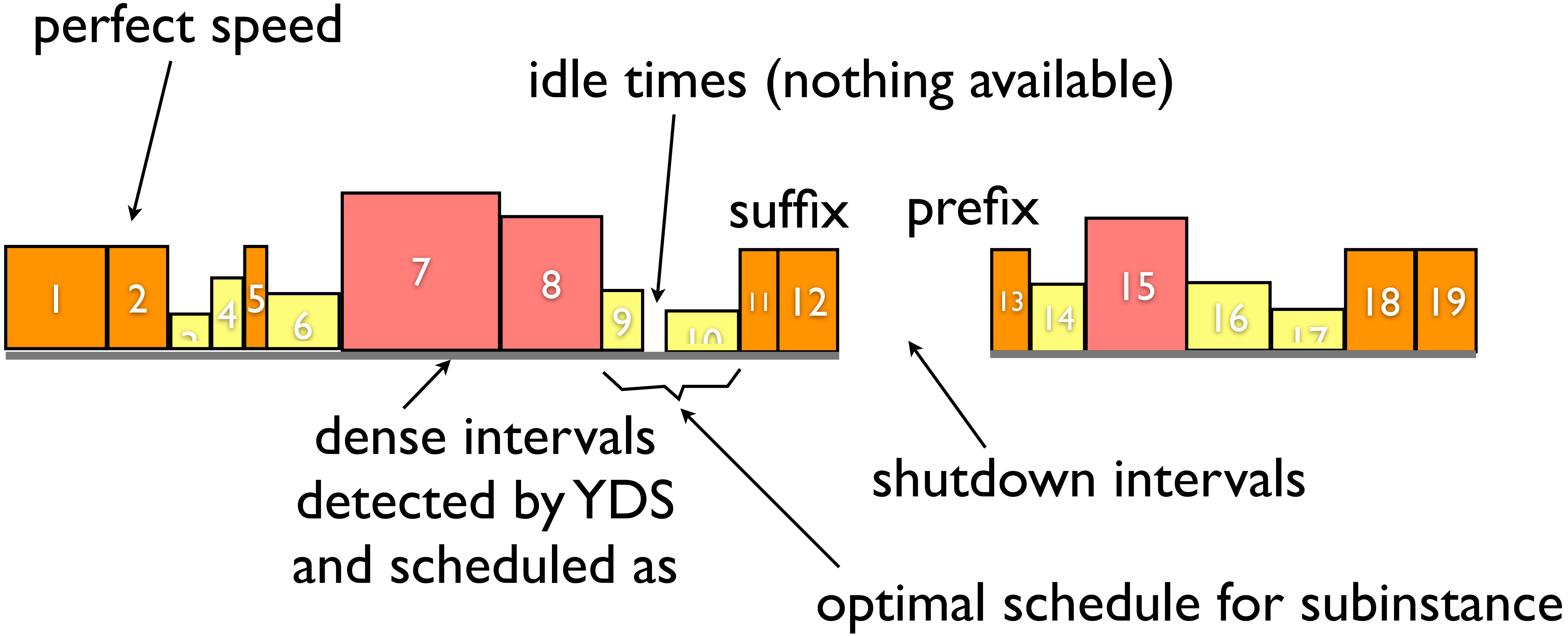
Speed scaling power down scheduling for agreeable deadline instances

by dynamic programming

Cost model

- wakeup cost = L
- ground energy = g
- scheduling at speed s, costs s^α+g per time unit, for some $2 \leq \alpha \leq 3$
- being idle costs g per time unit
- being shutdown costs nothing. But we pay L at wakeup

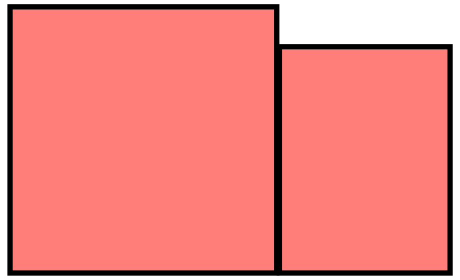
shape of optimal schedule



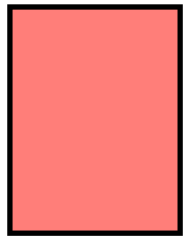
Dense intervals

- Definition perfect speed : speed s that minimizes $(1/s)(s^{\alpha}+g)$, which is the cost of scheduling one unit of workload at speed s. Ideally we would like to schedule only at this speed.
- compute speed scaling schedule without power down states (YDS)
- Intervals at perfect speed or more have to be scheduled like this
- They separate the remaining jobs into independent subinstances

initial subinstance



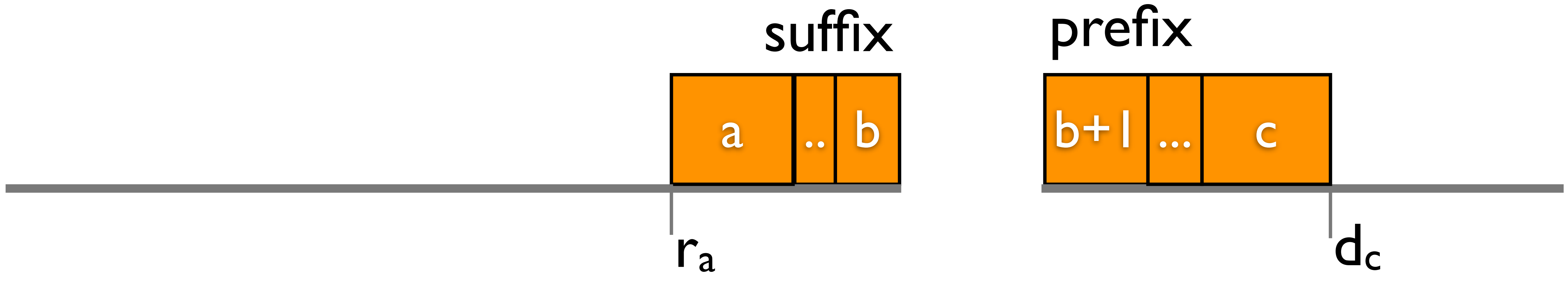
inner subinstances



final subinstance

Suffixes, prefixes

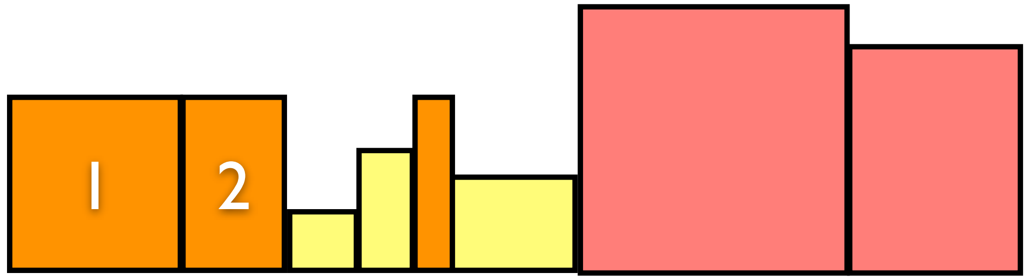
- Without loss of generality b is maximal (otherwise move job from prefix to suffix, without increasing cost)

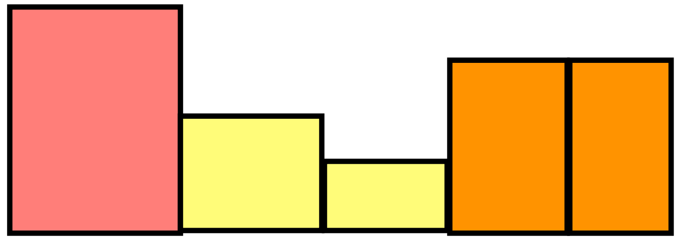


- A single left to right scan can compute all maximal suffix intervals [a,b] that partition jobs of a fixed subinstance

- A single right to left scan can compute a mapping, associating every job b+1 to the next job c=prefix(b+1), such that [b+1,c] forms a prefix

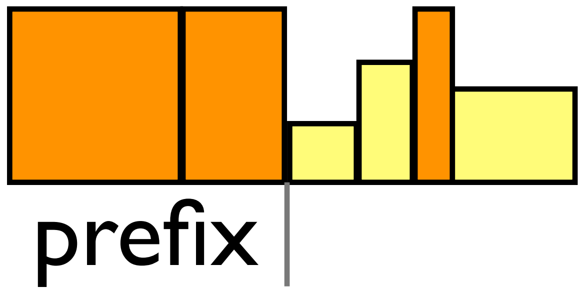
initial/final subinstances

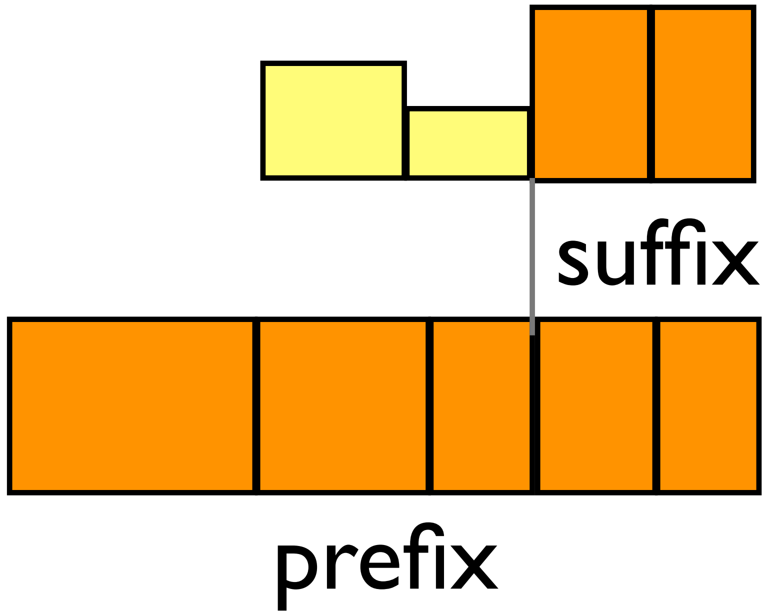




- If job 1 belongs to a dense interval, there is no initial subinstance.
- Same observation for job n and the final subinstance.
- the first prefix ranges from 1 to prefix(1)
  (unless job 1 is part of dense interval)
- the last suffix, is simply the last suffix interval computed

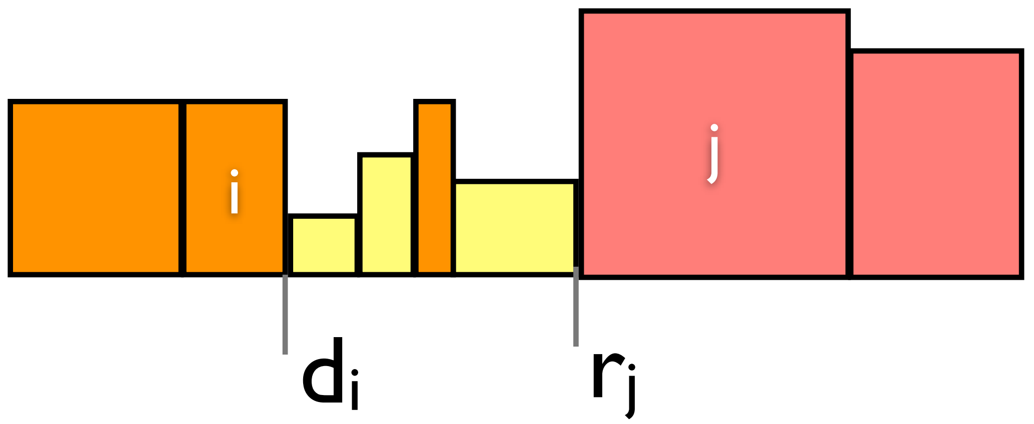
Subtle detail





- If there are no dense intervals, ...
- determine the last suffix.
- This defines an initial subinstance. As before compute prefix.
- If we are lucky the resulting inner subinstance is empty

Subinstance



- Every subinstance is defined by a job pair i,j
- consists of all jobs i+1,...,j-1, restricted to the intervals $[d_i,r_j]$
- is not defined (infinite cost) if some job is restricted to empty interval, i.e. if $d_{i+1}=d_i$ or $r_{j-1}=r_j$ or $d_i=r_j$

Implementation issue

- For the computation of Yij, we always have $1 \leq i < j \leq n$, and consider jobs $i+1,\dots,j-1$
- But for the computation of suffixes and prefixes, we might want to include first or last job. So we introduced dummy jobs 0 and n+1, with appropriate release times, deadlines:
- $d_0 = r_1$ and $r_{n+1} = d_n$

Dynamic program

- Y_{ij} optimal speed scaling without power down states schedule for subinstance (i,j)
- O_{ij} same, but allowing power down states and including ground energy

Base case

i

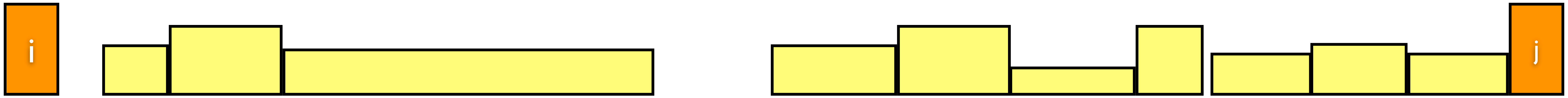
i+1

cost is $\min(L, (r_{i+1} - d_i) * g)$

shutdown

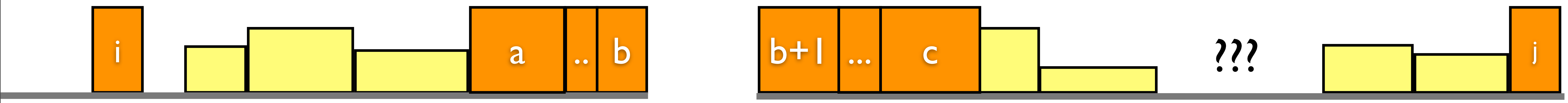
idle

Options



- If optimal schedule has no shutdown interval, its cost is $Y_{ij} + (r_{i+1} - d_i)*g$

Options

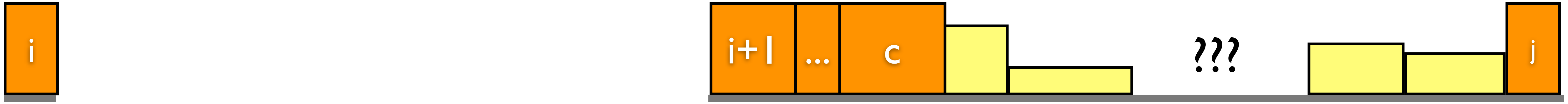


- If optimal schedule has a shutdown interval, and the first one is surrounded by jobs, then the suffix (a,b) is a maximal suffix and c=prefix(b+1)
- The cost is
  $Y_{ia} + (w_a+..+w_c)*(s^\alpha+g) + L + O_{cj}$

Options

- If optimal schedule has a shutdown interval, and the first one is not followed by jobs, then (a,j-1) is a maximal suffix
- The cost is
  $Y_{ia} + (w_a+..+w_j)*(s^{\alpha}+g) + L$

Options



- If optimal schedule has a shutdown interval, and the first one is not preceded by jobs, then the first prefix is (i+1,c) for c=prefix(i+1) and c<j
- The cost is
  $L + (w_{i+1}+..+w_c)*(s^{\alpha}+g) + O_{cj}$

Subtle detail

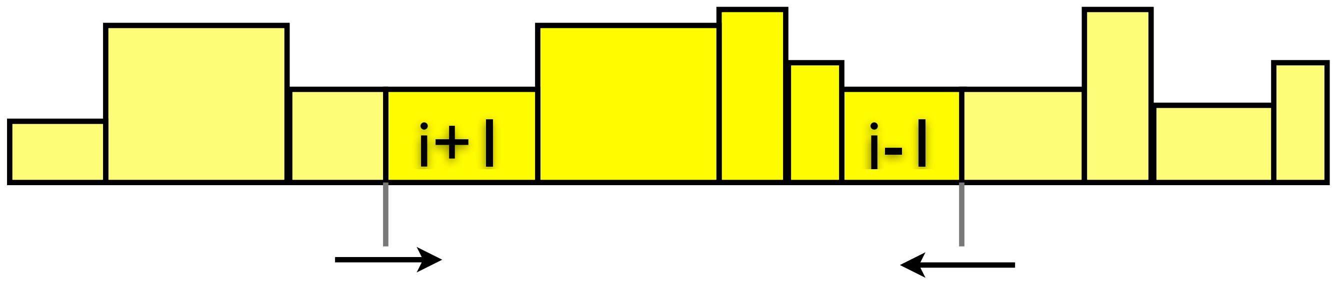
- The prefix does end at $\min(d_c, r_j)$
- Similarly, the suffix does start at $\max(r_a, d_i)$
- These boundaries have to be taken into account when computing suffixes and prefixes

Complexity

- $O(n^2)$ variables, each is computed by a minimization over $O(n)$ values
- For each of the $O(n^2)$ subinstances, computing the suffixes and the prefix map can be done in linear time by a simple one scan algorithm
- Computing Yij is the bottleneck.
  Here is how it can be done in time $O(n^3 \log n)$ ....

Computing optimal speedcaling without powerdown schedule

- First compute using the algorithm YDS the optimal schedule Y_{1n} for the whole instance.
- Then for each subinstance ij, "squeeze" the jobs i+1 and j-1.



- Left squeeze: if $d_i=d_{i+1}$, return infinite cost schedule. Otherwise move gradually start of schedule to d_i, according to an event priority list ...

Events

- merge event : time when first block has same speed as second block. Then generate split events for the jobs belonging now to the first block.

- split event : time when a job in the first block hits its deadline. Then split the block into two.

- processing : there will be a sequence of merge events, followed by sequence of split events. Each has linear length. Complexity : O(n log n)