Processor Sharing Scheduling with Linear Slowdown

Yoni Nazarathy*
The University of Queensland

Joint work with
Liron Ravner**, Moshe Haviv and Hai Le Vu

QANZIAM,
October, 2013,
Brisbane, Australia

* Work partially supported by ARC through DP130100156 and DE130100291

** With thanks to the Australia-Israel Scientific Exchange Foundation

Outline

- Model
- Dynamics
- Optimization

The model:
Processor sharing scheduling with linear slow down

The model

Background: "classic" processor sharing queue with N users

$$v\big(q(t)\big) = \frac{\beta}{q(t)},$$

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \qquad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

Think: CPU time-sharing batch jobs

The model

Background: "classic" processor sharing queue with N users

$$v\big(q(t)\big) = \frac{\beta}{q(t)},$$

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \qquad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

Think: CPU time-sharing batch jobs

In our model, set $v(\cdot)$ to have "linear slowdown"

$$v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big)$$

$\beta \equiv$ free flow speed
$\alpha \equiv$ slowdown rate
Assume $\beta - \alpha(N-1) > 0$

Think: aggregated urban road network

The model

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big) dt, \quad v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \quad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

The model

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \quad v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \quad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

Objective function for scheduling

$$c_i(a_i, d_i) = \gamma_1(d_i - d_i^*)^2 + \gamma_2(a_i - a_i^*)^2 + \gamma_3(d_i - a_i)$$

$$\text{TotalCost}\Big(a_1, \ldots, a_N\Big) = \sum_{i=1}^{N} c_i\big(a_i, d_i(a_1, \ldots, a_N)\big)$$

The model

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \quad v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \quad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

Objective function for scheduling

$$c_i(a_i, d_i) = \gamma_1(d_i - d_i^*)^2 + \gamma_2(a_i - a_i^*)^2 + \gamma_3(d_i - a_i)$$

$$\text{TotalCost}\Big(a_1, \ldots, a_N\Big) = \sum_{i=1}^{N} c_i\big(a_i, d_i(a_1, \ldots, a_N)\big)$$

Research goal: optimization of total cost

Dynamics:
How does $\{a_i\}$ determine $\{d_i\}$?

Dynamics

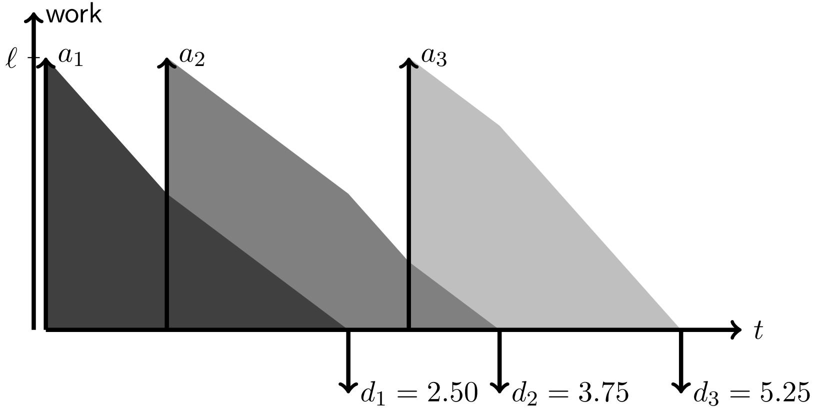
Example

$\beta = 15$, $\alpha = 5$, $N = 3$, $\ell_1 = \ell_2 = \ell_3 = 30$
In this case, "free flow travel time" $= 2$
Assume scheduler decides $a_1 = 0$, $a_2 = 1$ and $a_3 = 3$

Dynamics

Example

$\beta = 15$, $\alpha = 5$, $N = 3$, $\ell_1 = \ell_2 = \ell_3 = 30$
In this case, "free flow travel time" $= 2$
Assume scheduler decides $a_1 = 0$, $a_2 = 1$ and $a_3 = 3$



Dynamics

N equations

$$\ell_i = \int_{a_i}^{d_i} \beta - \alpha\Big(\big(\sum_{j=1}^{N} \mathbf{1}\{t \in [a_j, d_j]\}\big) - 1\Big) dt, \qquad i = 1, \ldots, N$$

Observation: If $\ell_i \equiv \ell$ then the arrival and departure sequences have the same order. **We make this assumption throughout**!

Dynamics

N equations

$$\ell_i = \int_{a_i}^{d_i} \beta - \alpha\Big(\big(\sum_{j=1}^{N} \mathbf{1}\{t \in [a_j, d_j]\}\big) - 1\Big)dt, \qquad i = 1, \ldots, N$$

Observation: If $\ell_i \equiv \ell$ then the arrival and departure sequences have the same order. **We make this assumption throughout**!

Label the arrivals: $a_1 \le a_2 \le \ldots \le a_N$
Thus the departures satisfy: $d_1 \le d_2 \le \ldots \le d_N$

Dynamics

N equations

$$\ell_i = \int_{a_i}^{d_i} \beta - \alpha\Big(\big(\sum_{j=1}^{N} \mathbf{1}\{t \in [a_j, d_j]\}\big) - 1\Big) dt, \qquad i = 1, \ldots, N$$

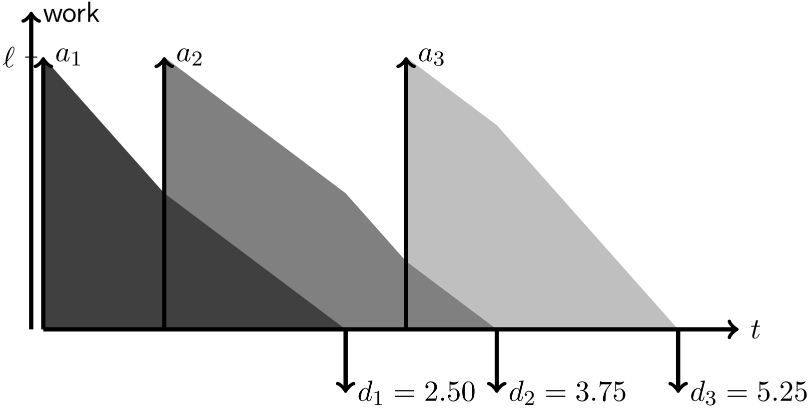
Observation: If $\ell_i \equiv \ell$ then the arrival and departure sequences have the same order. **We make this assumption throughout**!

Label the arrivals: $a_1 \leq a_2 \leq \ldots \leq a_N$
Thus the departures satisfy: $d_1 \leq d_2 \leq \ldots \leq d_N$

To describe the order of $\{a_i\}$ vs. $\{d_i\}$ use:

$$k_i := \max\big\{k \ : \ a_k \leq d_i\big\}, \quad \text{or} \quad h_i := \min\big\{h \ : \ d_h \geq a_i\big\}$$

Dynamics



$k_i := \max\{k \ : \ a_k \le d_i\}$, or $h_i := \min\{h \ : \ d_h \ge a_i\}$

In this example: $a_1 \le a_2 \le d_1 \le a_3 \le d_2 \le d_3$, so

$$k_1 = 2,\ k_2 = 3,\ k_3 = 3, \quad \text{and} \quad h_1 = 1,\ h_2 = 1,\ h_3 = 2$$

Determining $\{d_i\}$ based on $\{a_i\}$

Lemma

Assume that $\{a_i\}$ *are ordered:* $a_1 \leq a_2 \leq \ldots \leq a_N$ *and assume that* $k_1, \ldots, k_N$ *and* $h_1, \ldots, h_N$ *describe the order of* $\{d_i\}$*, then for* $i = 1, \ldots, N$*,*

$$d_i = \frac{\ell + \big(\beta - \alpha(i - h_i)\big)a_i + \alpha\Big(\sum_{j=h_i}^{i-1} d_j - \sum_{j=i+1}^{k_i} a_j\Big)}{\beta - \alpha(k_i - i)}$$

with the special cases,

$$d_1 = \frac{\ell + \beta a_1 - \alpha \sum_{j=2}^{k_1} a_j}{\beta - \alpha(k_1 - 1)}, \qquad d_N = \frac{\ell + \beta a_N + \alpha \sum_{j=h_N}^{N-1} (d_j - a_N)}{\beta}$$

Derivation

Simply "play around" with the N equations

$$\ell_i = \int_{a_i}^{d_i} \beta - \alpha\Big(\big(\sum_{j=1}^{N} \mathbf{1}\{t \in [a_j, d_j]\}\big) - 1\Big)dt, \qquad i = 1, \ldots, N$$

$$\begin{aligned}
\ell &= (\beta+\alpha)(d_i - a_i) - \alpha \sum_{j=1}^{N} \int_{a_i}^{d_i} \mathbf{1}\{t \in [a_j, d_j]\}dt \\
&= (\beta+\alpha)(d_i - a_i) - \alpha \sum_{j=1}^{N} \big(d_i \wedge d_j - a_i \vee a_j\big)^+ \\
&= (\beta+\alpha)(d_i - a_i) - \alpha \sum_{j=1}^{i-1} \big(d_i \wedge d_j - a_i \vee a_j\big)^+ - \alpha(d_i - a_i) - \alpha \sum_{j=i+1}^{N} \big(d_i \wedge d_j - a_i \vee a_j\big)^+ \\
&= -\beta a_i + (\beta - \alpha(N-i))d_i - \alpha \sum_{j=1}^{i-1} d_j + \alpha \sum_{j=1}^{i-1} (a_i \wedge d_j) + \alpha \sum_{j=i+1}^{N} (a_j \wedge d_i).
\end{aligned}$$

Now use k_i and h_i values to resolve sums with minimum...

Proposition

We have an algorithm that finds the unique $\{d_i\}$ corresponding to $\{a_i\}$ and requires at most $2N$ steps

Input: $\mathbf{a} = (a_1, ..., a_N)$
Output: $\mathbf{d} = (d_1, ..., d_N)$, $\mathbf{k} = (k_1, ..., k_N)$ and $\mathbf{h} = (h_1, ..., h_N)$

init $\mathbf{k} = \mathbf{h} = (1, 2, 3, \ldots, N)$
init $\mathbf{d} = \emptyset$
for $i = 1, \ldots, N$ **do**
  init $k = i \vee k_{i-1}$
  compute $\tilde{d}_i(k, h_i, \mathbf{d})$
  while $\tilde{d}_i(k, h, \mathbf{d}) \leq a_{k+1}$ **do**
    increment k
    compute $\tilde{d}_i(k, h_i, \mathbf{d})$
  end while
  set $k_i = k$
  set $d_i = \tilde{d}_i(k, h_i, \mathbf{d})$
  set $h_{i+1} = \tilde{h}_{i+1}(k_1, \ldots, k_{i+1})$
end for
return $(\mathbf{d}, \mathbf{k}, \mathbf{h})$

Towards optimization procedures

Supporting algorithms for optimization

We have the following algorithms

1. Optimizing efficiently over one coordinate (changing one a_i) and keeping the rest fixed
2. More generally a line search for optimizing over some line
3. Given an ordering (e.g. $k_1, \ldots, k_N$), a specification of a quadratic program for optimizing over a region of a's that maintain that order

Supporting algorithms for optimization

We have the following algorithms

1. Optimizing efficiently over one coordinate (changing one a_i) and keeping the rest fixed
2. More generally a line search for optimizing over some line
3. Given an ordering (e.g. $k_1, \ldots, k_N$), a specification of a quadratic program for optimizing over a region of a's that maintain that order

Our goal is to have an optimization procedure that uses the above and attains a local minimum efficiently (perhaps with a guaranteed probability)

We have the following algorithms

1. Optimizing efficiently over one coordinate (changing one a_i) and keeping the rest fixed
2. More generally a line search for optimizing over some line
3. Given an ordering (e.g. $k_1, \dots, k_N$), a specification of a quadratic program for optimizing over a region of a's that maintain that order

Our goal is to have an optimization procedure that uses the above and attains a local minimum efficiently (perhaps with a guaranteed probability)

We are still not fully there!

Possible naive optimization procedures

Naive uses of the supporting algorithms

1. Optimizing efficiently over one coordinate (changing one a_i) and keeping the rest fixed
   $\rightarrow$ **Fits naturally in "coordinate pivot iteration" (but our objective is not smooth)!**

Possible naive optimization procedures

Naive uses of the supporting algorithms

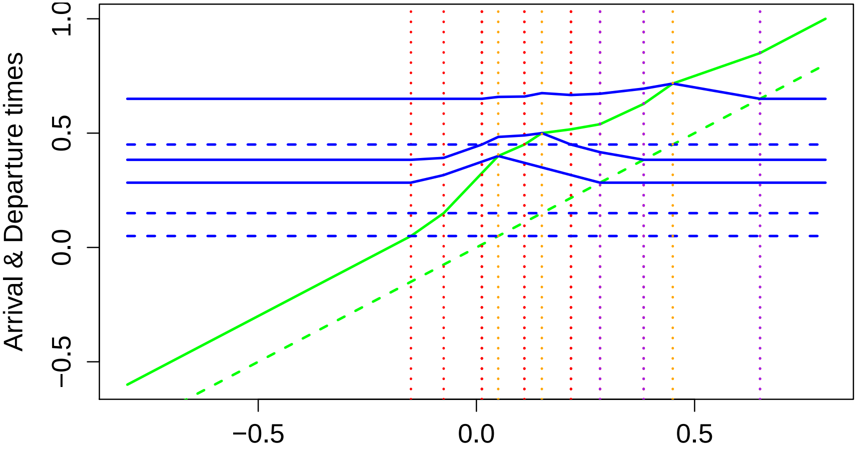
1. Optimizing efficiently over one coordinate (changing one a_i) and keeping the rest fixed
   $\rightarrow$ **Fits naturally in "coordinate pivot iteration" (but our objective is not smooth)!**
2. More generally a line search for optimizing over some line
   $\rightarrow$ **Can also be used in an iteration procedure over N orthogonal directions**

Possible naive optimization procedures

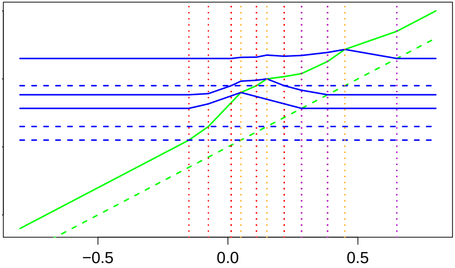
Naive uses of the supporting algorithms

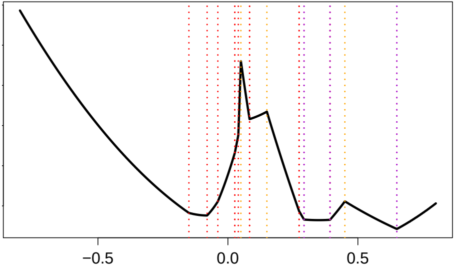
1. Optimizing efficiently over one coordinate (changing one a_i) and keeping the rest fixed
   $\rightarrow$ **Fits naturally in "coordinate pivot iteration" (but our objective is not smooth)!**
2. More generally a line search for optimizing over some line
   $\rightarrow$ **Can also be used in an iteration procedure over N orthogonal directions**
3. Given an ordering (e.g. $k_1, \ldots, k_N$), a specification of a quadratic program for optimizing over a region of a's that maintain that order
   $\rightarrow$ **Is useful for exhaustive search in finite time**

Optimizing over the arrival of one user (a_i)



Optimizing over the arrival of one user (a_i)





Optimizing over the arrival of one user (a_i)

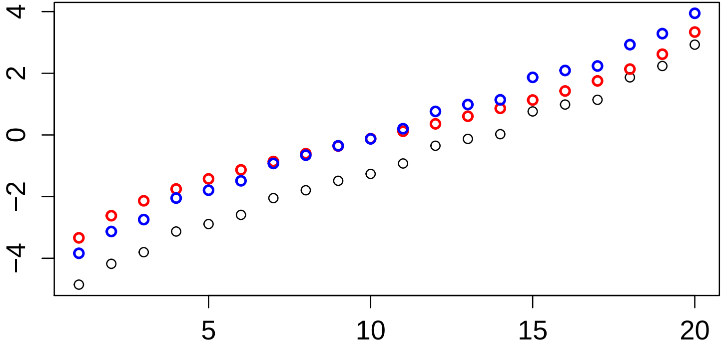
```
init a = sort(a^(r)), x = a̲
run Alg. 1(a) → (d, k, h)
init a_*^(r) = a^(r), m_*^(r) = T(x)
while x ≤ ā do
    init π = order(a^(r)), a = sort(a_{-r} ∪ x)
    compute θ, η, 𝒯, t, T'(x) > 0
    if T'(x)^(r) < 0 then
        compute x_0 and T(x_0)
        if x_0 < x + t then
            if T(x_0) < m_*^(r) then
                set a_*^(r) = (a_*^(r))_{-r} ∪ (x_0), m_*^(r) = T(x_0)
            end if
        else if T(x + t) < m_*^(r) then
            set a_*^(r) = (a_*^(r))_{-r} ∪ (x + t), m_*^(r) = T(x + t)
        end if
    end if
    set x = x + t
    for τ ∈ 𝒯 do
        if τ ∈ {1, . . . , N then
            if θ_τ < 0 then
                h_{k_τ} = h_{k_τ} + 1, k_τ = k_τ − 1
            else if θ_τ > 0 then
                k_τ = k_τ + 1, h_{k_τ} = h_{k_τ} − 1
            end if
        end if
    end for
end while
return a_*^(r) and m_*^(r)
```

Each one coordinate search is efficient

Proposition

In any execution of a one coordinate search, time is broken up to at most $\frac{1}{6}N^3 - \frac{1}{2}N^2 + \frac{7}{6}N$ *intervals*

Optimization example (coordinate pivot iteration)

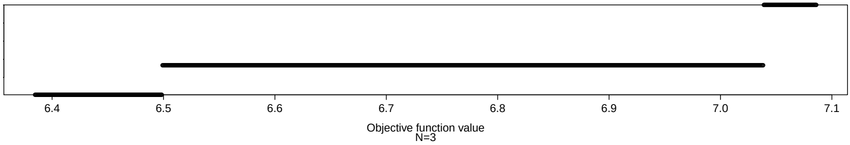


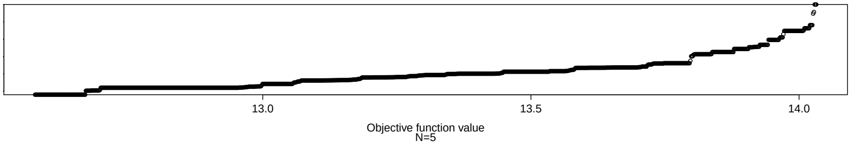
d_i^* – desired departure time

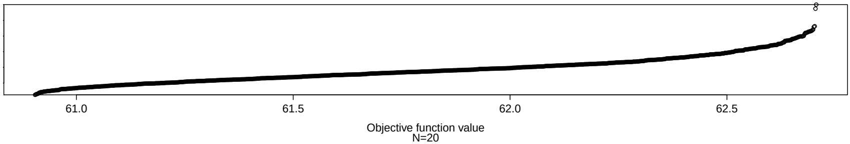
a_i – optimized arrival time

d_i – actual departure time

Convergence **not** guaranteed!







Wrap up

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \quad v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \quad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

$$c_i(a_i, d_i) = \gamma_1(d_i - d_i^*)^2 + \gamma_2(a_i - a_i^*)^2 + \gamma_3(d_i - a_i)$$

$$\mathsf{TotalCost}\Big(a_1, \ldots, a_N\Big) = \sum_{i=1}^{N} c_i\big(a_i, d_i(a_1, \ldots, a_N)\big)$$

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \quad v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \quad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

$$c_i(a_i, d_i) = \gamma_1 (d_i - d_i^*)^2 + \gamma_2 (a_i - a_i^*)^2 + \gamma_3 (d_i - a_i)$$

$$\text{TotalCost}\Big(a_1, \ldots, a_N\Big) = \sum_{i=1}^{N} c_i\big(a_i, d_i(a_1, \ldots, a_N)\big)$$

Summary

- Efficient algorithm for local minimum – **Not yet**
- Finite time algorithm for global minimum – **Yes**
- Outlook — **???**

$$\ell_i = \int_{a_i}^{d_i} v\big(q(t)\big)dt, \quad v\big(q(t)\big) = \beta - \alpha\big(q(t) - 1\big), \quad q(t) = \sum_{i=1}^{N} \mathbf{1}\{t \in [a_i, d_i]\}$$

$$c_i(a_i, d_i) = \gamma_1(d_i - d_i^*)^2 + \gamma_2(a_i - a_i^*)^2 + \gamma_3(d_i - a_i)$$

$$\mathsf{TotalCost}\Big(a_1, \ldots, a_N\Big) = \sum_{i=1}^{N} c_i\big(a_i, d_i(a_1, \ldots, a_N)\big)$$

Summary

- Efficient algorithm for local minimum – **Not yet**
- Finite time algorithm for global minimum – **Yes**
- Outlook — **???**

Thanks and enjoy your lunch!!!