An Asynchronous Distributed Algorithm for Solving Stochastic Unit Commitment at an Industrial Scale

Ignacio Aravena and Anthony Papavasiliou

Prepared for presentation at,
FERC Technical Conference 2017,
Washington DC, USA. June 27th, 2017.

Outline

1. Motivation
2. Preliminaries
3. Asynchronous distributed block-coordinate subgradient method for dual minimization
4. Primal recovery
5. High performance computing implementation
6. Numerical results
   - □ Western Electricity Coordinating Council (WECC) system
   - □ Central Western European (CWE) system
7. Conclusions

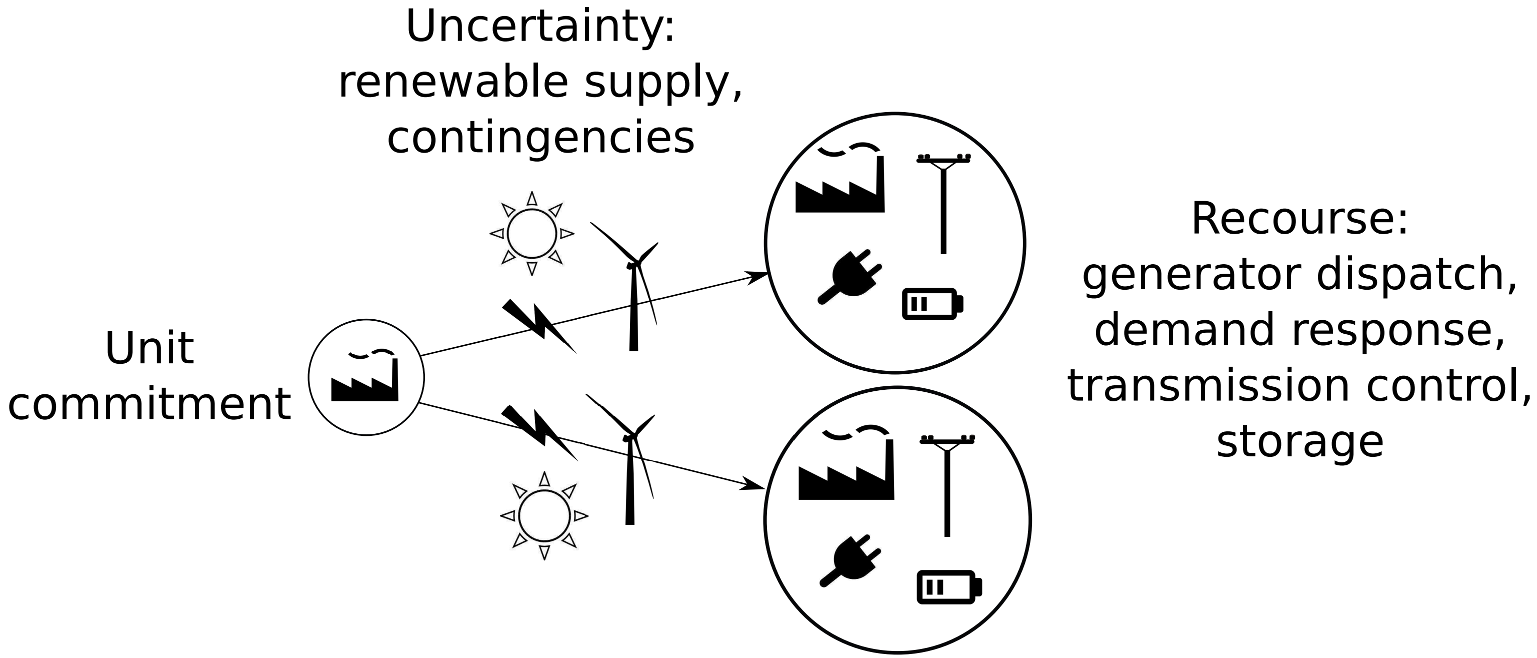
Motivation

Motivation

- Renewable energy integration levels rising worldwide following environmental targets
- Stochastic unit commitment offers advantages over deterministic reserve policies for coping with uncertainty (Takriti and Birge, 1996), (Papavasiliou and Oren, 2013)
- Stochastic unit commitment has failed to become an industry standard:
  - Market design compatible with treatment of uncertainty
  - Difficulty and scale of stochastic unit commitment models
- Decomposition and parallelization have shown promise to solve stochastic unit commitment models (Cheung et al., 2015), (Kim and Zavala, 2015)

Stochastic unit commitment problem

□ Stochastic unit commitment explicitly models uncertainty in power system operations



Uncertainty: renewable supply, contingencies

Unit commitment

Recourse: generator dispatch, demand response, transmission control, storage

Stochastic unit commitment problem

□ The two-stage stochastic unit commitment problem can be formulated as

$$\begin{aligned} \max_{u,v,w} \quad & \sum_{i=1}^{N} (\boldsymbol{c}_i^T \boldsymbol{v}_i + \boldsymbol{d}_i^T \boldsymbol{w}_i) \\ \text{s.t.} \quad & \boldsymbol{v}_i - \boldsymbol{u} = \boldsymbol{0}, \quad i = 1, \ldots, N \\ & (\boldsymbol{v}_i, \boldsymbol{w}_i) \in \mathcal{D}_i, \quad i = 1, \ldots, N \\ & \boldsymbol{u} \in \mathcal{U} \end{aligned}$$

– $\boldsymbol{u}$ corresponds to non-anticipative commitment and production variables of generators

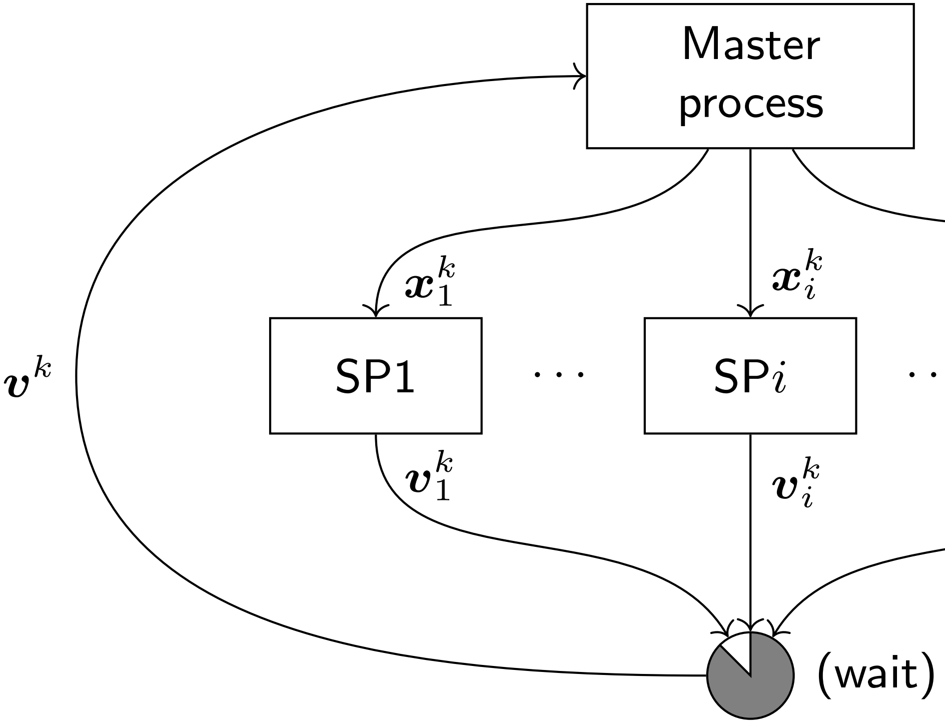
– $\boldsymbol{v}_i$ are local copies of $\boldsymbol{u}$ for each scenario i

– $\boldsymbol{w}_i$ are commitment, production and transmission recourse variables

□ Scenario decomposition relax non-anticipativity constraints in order to compute a solution iteratively

Synchronous parallel decomposition method

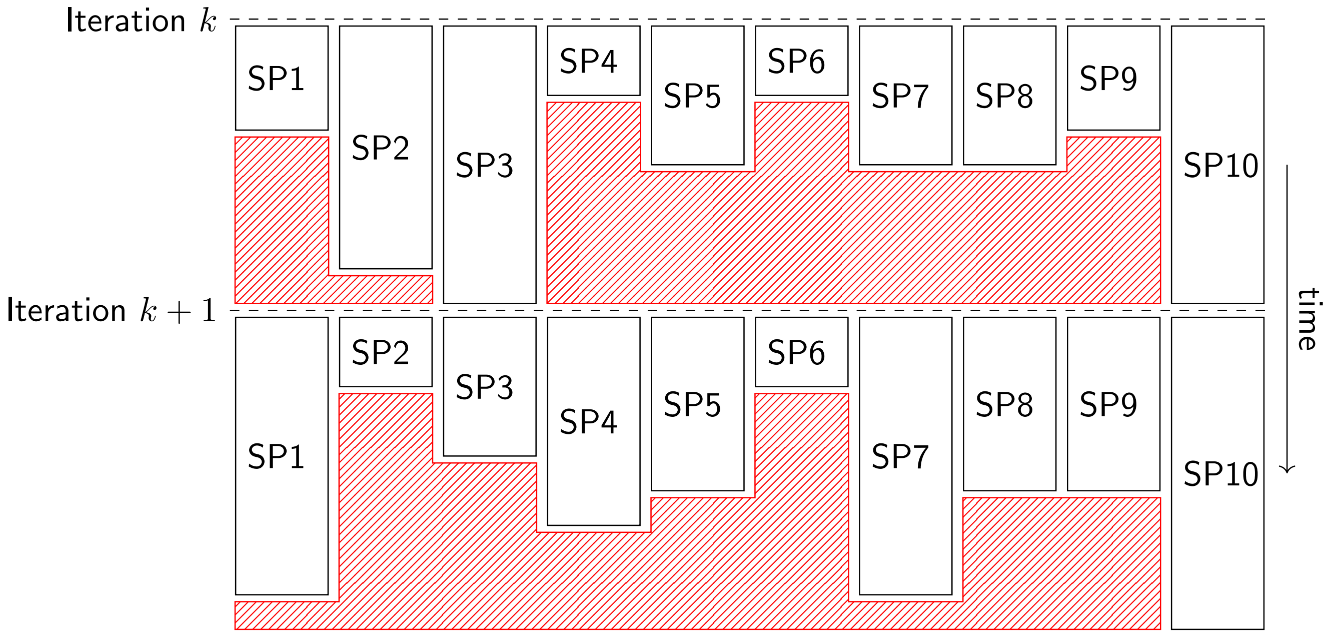
□ Parallelization of common decomposition methods leads to synchronous parallel algorithms, e.g. (Papavasiliou and Oren, 2013), (Cheung et al., 2015)



Execution of a synchronous parallel decomposition method.

Synchronous parallel decomposition method

Load profile of a synchronous parallel decomposition method. Red area indicates idle processors.



□ Evaluation times of subproblems (SPi) can vary significanlty:

- Up to 48x across subproblems within the same iteration
- Up to 25x across iterations for the same subproblem

Contributions of the present work

- □ We develop an **asynchronous distributed algorithm** for stochastic unit commitment which fully harness the power of distributed computing infrastructure

- □ We use scenario decomposition and propose an asynchronous block-coordinate subgradient method for minimizing the Lagrangian dual of stochastic unit commitment (convex, non-differentiable).

  The proposed method does not perform a line search along coordinates on each iteration (Tseng and Yun, 2009), (Fercoq and Richtárik, 2013).

- □ We propose primal recovery heuristics and implement the proposed asynchronous algorithm on a high performance computing cluster. The algorithm is able to solve industry-scale instances within the same time frame of deterministic unit commitment.

Preliminaries

Preliminaries: Stochastic unit commitment

□ The stochastic unit commitment problem

$$\begin{aligned}
\max_{u,v,w} \quad & \sum_{i=1}^{N} (\boldsymbol{c}_i^T \boldsymbol{v}_i + \boldsymbol{d}_i^T \boldsymbol{w}_i) \\
\text{s.t.} \quad & \boldsymbol{v}_i - \boldsymbol{u} = \boldsymbol{0}, \quad i = 1, \ldots, N \qquad (\boldsymbol{x}_i) \\
& (\boldsymbol{v}_i, \boldsymbol{w}_i) \in \mathcal{D}_i, \quad i = 1, \ldots, N \\
& \boldsymbol{u} \in \mathcal{U}
\end{aligned}$$

- $\mathcal{U}$ is a bounded convex set, described by linear constraints
- $\mathcal{D}_i$ is a bounded non-convex set, described by mixed integer-linear constraints
- $\boldsymbol{x}_i$ are dual multipliers associated to non-anticipativity constraints

Preliminaries: Scenario decomposition

- □ Lagrange relaxation of non-anticipativity constraints leads to the following separable dual problem

$$\min_{x \in \mathbb{R}^m} \quad f_0(\boldsymbol{x}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_i)$$

where f_0 and f_i are defined according to

$$f_0(\boldsymbol{x}) := \sup_{u \in \mathcal{U}} \left(-\sum_{i=1}^{N} {\boldsymbol{x}_i}^T \right) \boldsymbol{u}$$

$$f_i(\boldsymbol{x}_i) := \sup_{(v,w) \in \mathcal{D}_i} \left((\boldsymbol{c}_i^T + {\boldsymbol{x}_i}^T)\boldsymbol{v} + \boldsymbol{d}_i^T \boldsymbol{w} \right) \quad i = 1, \ldots, N$$

- □ Evaluation of $\boldsymbol{f_0}$ **requires to solve an LP**, while evaluation of $\boldsymbol{f_i}$ **involves solving a large-scale MILP**

Preliminaries: Smooth approximation of f_0

□ In order to obtain convergence guarantees, following (Nesterov, 2005), (Fercoq and Richtárik, 2013), we replace the non-decomposable part of the objective f_0 by the following smooth approximation

$$f_0^{\mu}(\boldsymbol{x}) := \sup_{u \in \mathcal{U}} \left(\left(-\sum_{i=1}^{N} \boldsymbol{x}_i^T \right) \boldsymbol{u} - \frac{1}{2}\mu \|\boldsymbol{u} - \boldsymbol{u}_0\|_2^2 \right)$$

□ f_0^{μ} is a differentiable function with Lipchitz gradient with constant L_0^{μ}

□ We focus then on minimizing the following approximation of the dual problem of the stochastic unit commitment

$$\min_{x \in X} \; f(\boldsymbol{x}) = f_0^{\mu}(\boldsymbol{x}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_i)$$

Asynchronous distributed block-coordinate subgradient method

Serial block-coordinate subgradient method

$$\min_{x \in X} \; f(\boldsymbol{x}) = f_0^\mu(\boldsymbol{x}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_i)$$

□ Consider the randomized coordinate descent method (Nesterov, 2012):

1. Let $k := 0, \boldsymbol{x}^k := \bar{\boldsymbol{x}}^0$
2. Select component $j(k)$ uniformly at random from $\{1, \ldots, N\}$
3. Compute $\nabla f_0^\mu(\boldsymbol{x}^k)$ and $g(j(k), \boldsymbol{x}^k_{j(k)}) \in \partial f_{j(k)}(\boldsymbol{x}^k_{j(k)})$
4. Perform update according to

$$\boldsymbol{x}^{k+1} := \mathcal{P}_X \left[\boldsymbol{x}^k - \lambda_k \cdot I^T_{j(k)} \Big(I_{j(k)} \nabla f_0^\mu(\boldsymbol{x}^k) + g\big(j(k), \boldsymbol{x}^k_{j(k)}\big) \Big) \right]$$

5. Let $k := k + 1$ and return to 2.

Convergence of the serial method

- □ Consider the expected update direction of the serial method,

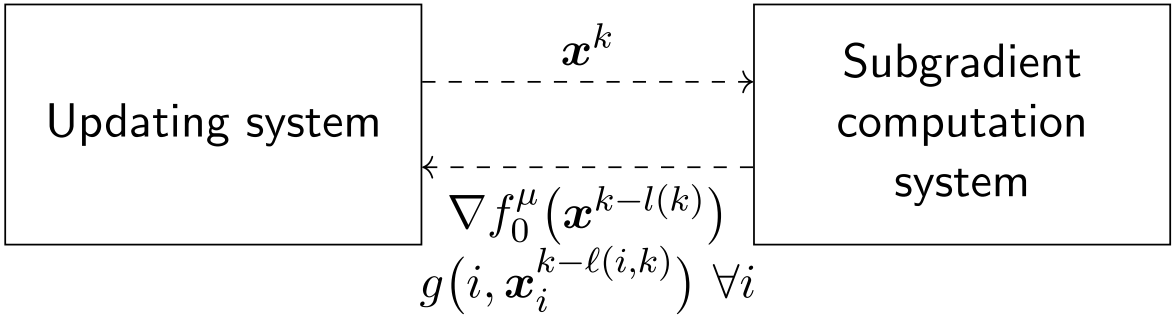
$$\mathbb{E}[I_J^T(I_J\nabla f_0^\mu(\boldsymbol{x}^k) + g(J, \boldsymbol{x}_J^k))|\,\boldsymbol{x}^k],$$

  where J is a discrete uniformly distributed variable on $\{1, \ldots, N\}$.

- □ The expected update direction coincides with the direction of a subgradient of f at $\boldsymbol{x}^k$
  - – This property requires f_0^μ to be smooth
  - – The serial method is equivalent to the **stochastic subgradient method** (Ermoliev, 1983)

- □ Provided we choose a diminishing, non-summable and square summable stepsize λ_k, the algorithm will converge to an optimal solution with probability 1.

Asynchronous method: computation model



Updating system

$\boldsymbol{x}^k$

Subgradient computation system

$\nabla f_0^{\mu}\big(\boldsymbol{x}^{k-l(k)}\big)$

$g\big(i, \boldsymbol{x}_i^{k-\ell(i,k)}\big)\ \forall i$

Conceptual computation model (Nedić et al., 2001).

□ The *Subgradient computation system* evaluates ∇f_0^{μ} and subgradients of component functions in parallel

□ The *Updating system* stores the last gradient evaluated $\nabla f_0^{\mu}(\boldsymbol{x}^{k-l(k)})$ and the latest subgradients evaluated $g(i, \boldsymbol{x}^{k-\ell(i,k)})\ \forall i$

□ Updates are performed using the lastly available information after each evaluation of a subgradient

□ Delays $l(k)$ and $\ell(i,k)$ appear because evaluation of gradients and subgradients is not instantaneous

Updating system operations

□ The *Updating system* performs the following operations:

1. Wait for next subgradient evaluation. Store new information when received.

2. Select component $j(k)$ uniformly at random from $\{1, \ldots, N\}$

3. Perform update according to

$$\boldsymbol{x}^{k+1} := \mathcal{P}_X \left[\boldsymbol{x}^k - \lambda_k \cdot I_{j(k)}^T \Big(I_{j(k)} \nabla f_0^{\mu} \big(\boldsymbol{x}^{k-l(k)} \big) + g\big(j(k), \boldsymbol{x}_{j(k)}^{k-\ell(j(k),k)} \big) \Big) \right]$$

4. Let $k := k + 1$ and return to 1.

Remark: Whenever $X = \mathbb{R}$, an update to block $j(k)$ increases $l(k)$ and $\ell(j(k), k)$ by one unit, but it does not affect $\ell(i, k)$ for $i \neq j(k)$

Convergence of the asynchronous method

- Under certain assumptions, we can show that the expected update direction of the asynchronous method is an approximate subgradient of f at $\boldsymbol{x}^k$

- In particular, the expected update direction $\mathbb{E}[I_J^T(I_J \nabla f_0^{\mu}(\boldsymbol{x}^{k-l(k)}) + g(J, \boldsymbol{x}_J^{k-\ell(j,k)}))| \mathcal{F}_k]$, where J is a discrete uniform random variable on $\{1, \ldots, N\}$ and $\mathcal{F}_k = \{\boldsymbol{x}^k, \boldsymbol{x}^{k-1}, \ldots, \boldsymbol{x}^0\}$, complies with

$$N \cdot (\boldsymbol{x}^k - \boldsymbol{y})^T \mathbb{E}\left[I_J^T \left(I_J \nabla f_0^{\mu}(\boldsymbol{x}^{k-l(k)}) + g\left(J, \boldsymbol{x}_J^{k-\ell(J,k)}\right)\right) \middle| \mathcal{F}_k \right] \geq$$

$$f(\boldsymbol{x}^k) - f(\boldsymbol{y}) - \left(C^2 L_0^{\mu} \sum_{m=k-L}^{k-1} \lambda_m^2 + 2CDN \sum_{m=k-L}^{k-1} \lambda_m \right)$$

- The asynchronous method can be shown to converge to an optimal solution of the dual problem with probability 1, see (Ermoliev, 1983) and (Nedić, 2002).

Upper bound computation

□ Unlike most asynchronous algorithm applications, we require computing bounds on the dual objective $\rightarrow$ **duality gap**

□ Recall we are trying to solve the following problem,

$$\min_{x \in \mathbb{R}^m} \quad f_0(\boldsymbol{x}) + \sum_{i=1}^{N} f_i(\boldsymbol{x}_i)$$

□ We can compute an upper bound, after each subgradient evaluation, at the cost of evaluating f_0 for a composite of the evaluated $\boldsymbol{x}_i$'s as follows

$$UB^k := f_0\left(\left[\left(\boldsymbol{x}_1^{k-\ell(1,k)}\right)^T \ldots \left(\boldsymbol{x}_N^{k-\ell(N,k)}\right)^T\right]^T\right) + \sum_{j=1}^{N} f_j\left(\boldsymbol{x}_j^{k-\ell(j,k)}\right)$$

Primal recovery

Primal recovery

□ We consider 4 methods for generating primal candidates:

1. First-in-first-out (**FIFO**)
   Evaluating solutions to scenario subproblems as they arrive

2. Random (**RND**)
   Selecting solutions to scenario subproblems at random

3. Last-in-first-out (**LIFO**)
   Evaluating solutions to scenario subproblems in reverse order

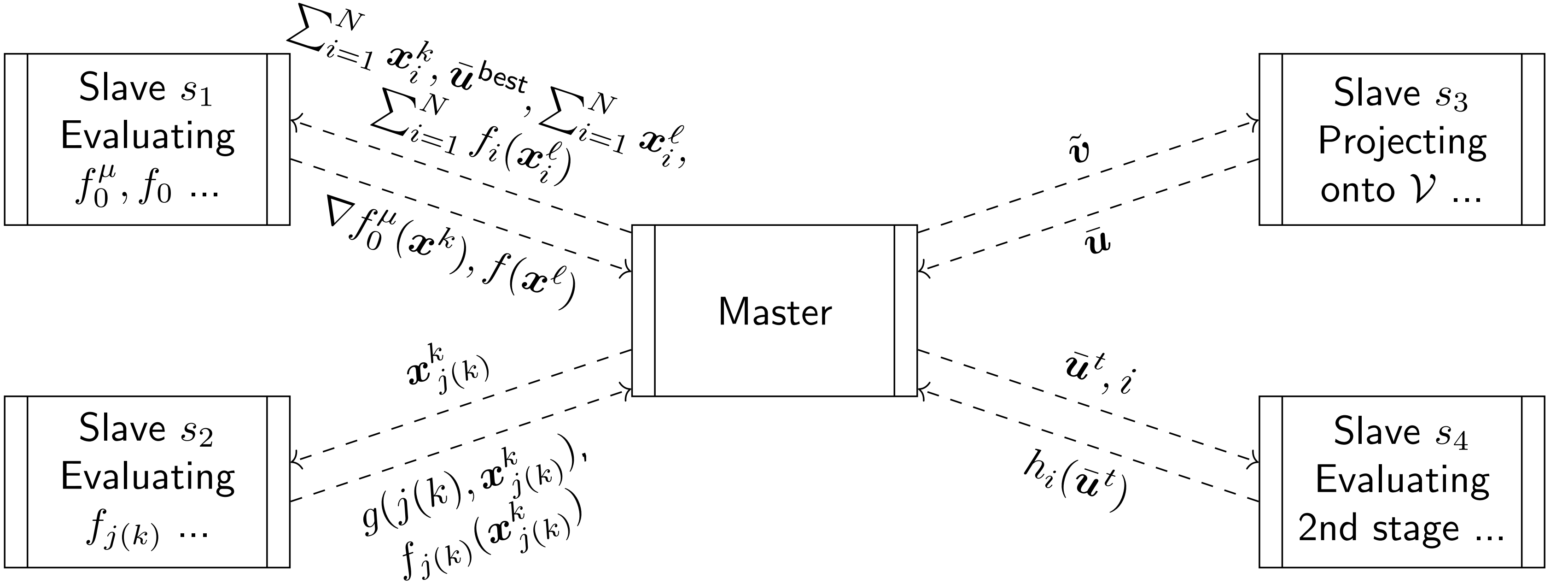
4. Importance sampling recombination heuristic (**IS**)
   Recall that $f_i(\boldsymbol{x}_i) := \sup_{(v,w)\in\mathcal{D}_i} \big((\boldsymbol{c}_i^T + \boldsymbol{x}_i^T)\boldsymbol{v} + \boldsymbol{d}_i^T\boldsymbol{w}\big)$ and let $\bar{\boldsymbol{v}}_i^*$ be the last solution to scenario subproblem $i, \forall i$

   (a) Pick a sample from $\{1, \dots, N\}$ based on the **estimated importance** of each scenario.

   (b) Average the $\bar{\boldsymbol{v}}_i^*$'s of the sample and project the result onto $\mathcal{V}$, i.e. the feasible set of $\boldsymbol{v}$.

High performance computing implementation

HPC implementation diagram



Slave s_1
Evaluating
f_0^μ, f_0 ...

$\sum_{i=1}^N \boldsymbol{x}_i^k, \bar{\boldsymbol{u}}^{\text{best}}, \sum_{i=1}^N \boldsymbol{x}_i^\ell,$
$\sum_{i=1}^N f_i(\boldsymbol{x}_i^\ell)$

$\nabla f_0^\mu(\boldsymbol{x}^k), f(\boldsymbol{x}^\ell)$

Master

$\tilde{\boldsymbol{v}}$

$\bar{\boldsymbol{u}}$

Slave s_3
Projecting
onto $\mathcal{V}$ ...

$\boldsymbol{x}^k_{\mathrm{j}(k)}$

$\mathrm{g}(\mathrm{j}(k), \boldsymbol{x}^k_{\mathrm{j}(k)}),$
$\mathrm{f}_{\mathrm{j}(k)}(\boldsymbol{x}^k_{\mathrm{j}(k)})$

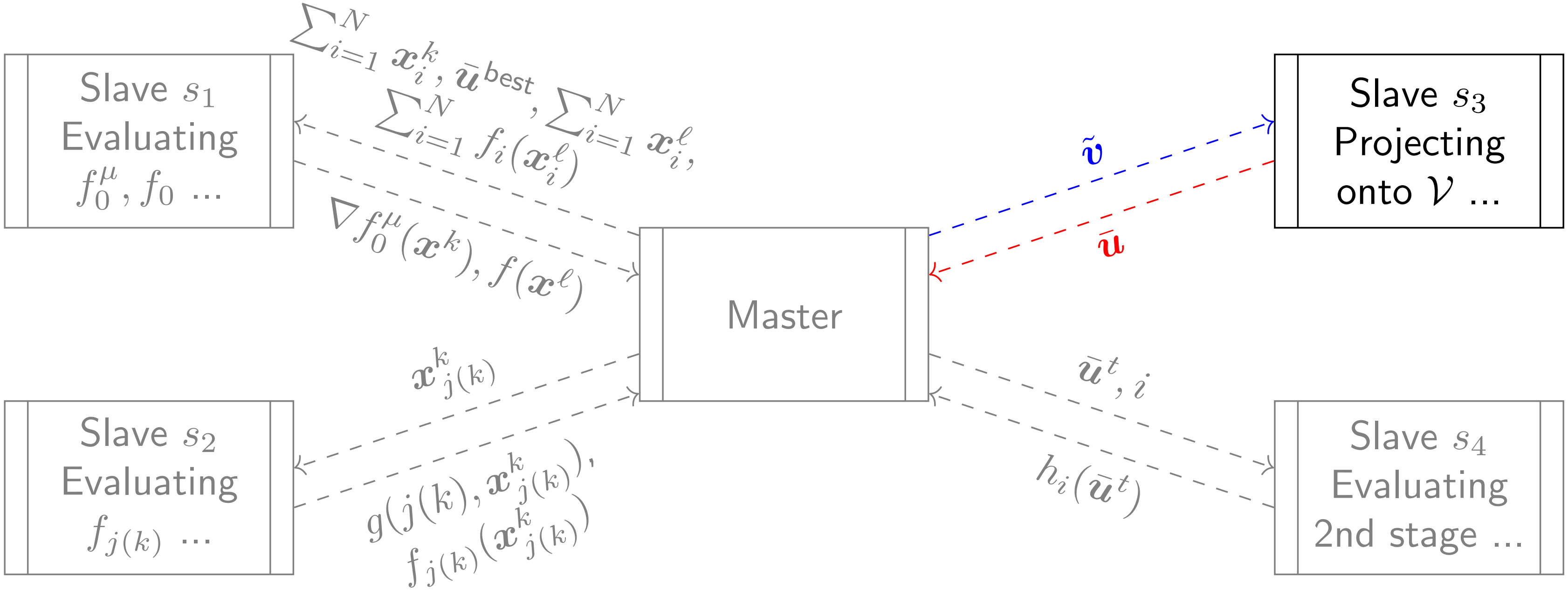
Slave s_2
Evaluating
$f_{j(k)}$ ...

$\bar{\boldsymbol{u}}^t, i$

$h_i(\bar{\boldsymbol{u}}^t)$

Slave s_4
Evaluating
2nd stage ...

Primal projection



Slave s_1

Evaluating

f_0^μ, f_0 ...

$\sum_{i=1}^N \boldsymbol{x}_i^k, \bar{\boldsymbol{u}}^{\text{best}}, \sum_{i=1}^N \boldsymbol{x}_i^\ell,$

$\sum_{i=1}^N f_i(\boldsymbol{x}_i^\ell)$

$\nabla f_0^\mu(\boldsymbol{x}^k), f(\boldsymbol{x}^\ell)$

$\tilde{\boldsymbol{v}}$

Slave s_3

Projecting

onto $\mathcal{V}$ ...

$\bar{\boldsymbol{u}}$

Master

$\boldsymbol{x}^k_{j(k)}$

Slave s_2

Evaluating

$f_{j(k)}$ ...

$g(j(k), \boldsymbol{x}^k_{j(k)}),$

$f_{j(k)}(\boldsymbol{x}^k_{j(k)})$

$\bar{\boldsymbol{u}}^t, i$

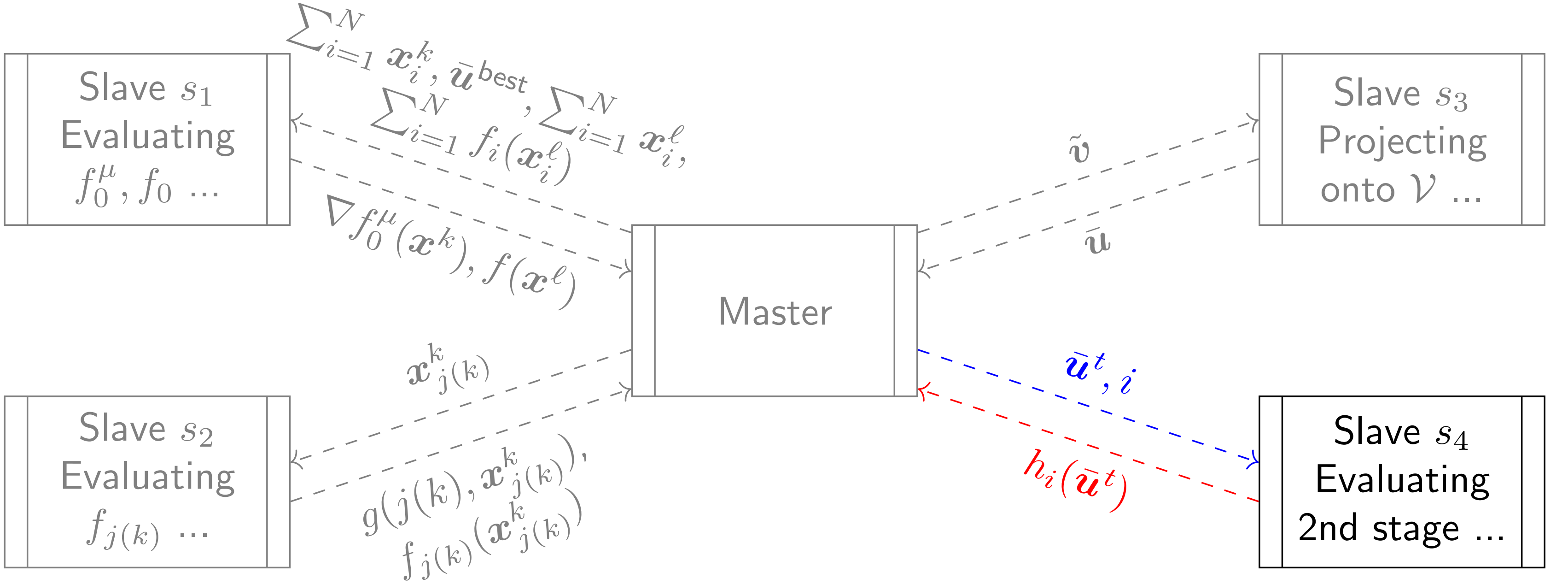
$h_i(\bar{\boldsymbol{u}}^t)$

Slave s_4

Evaluating

2nd stage ...

Second stage subproblem



Slave s_1
Evaluating
f_0^{μ}, f_0 ...

Slave s_3
Projecting
onto $\mathcal{V}$ ...

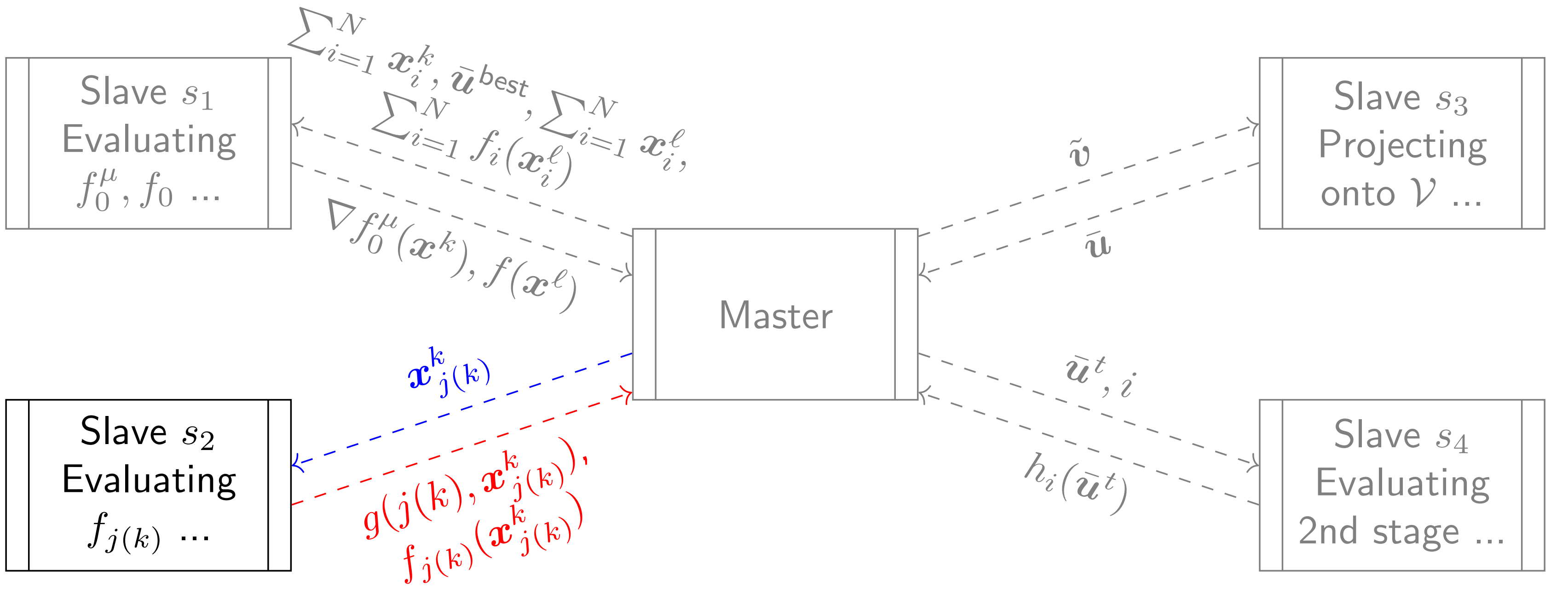
Master

Slave s_2
Evaluating
$f_{j(k)}$ ...

Slave s_4
Evaluating
2nd stage ...

$\sum_{i=1}^{N} \boldsymbol{x}_i^k, \bar{\boldsymbol{u}}^{\text{best}}, \sum_{i=1}^{N} \boldsymbol{x}_i^{\ell},$
$\sum_{i=1}^{N} f_i(\boldsymbol{x}_i^{\ell})$
$\nabla f_0^{\mu}(\boldsymbol{x}^k), f(\boldsymbol{x}^{\ell})$
$\tilde{\boldsymbol{v}}$
$\bar{\boldsymbol{u}}$
$\boldsymbol{x}^k_{j(k)}$
$g(j(k), \boldsymbol{x}^k_{j(k)}),$
$f_{j(k)}(\boldsymbol{x}^k_{j(k)})$
$\bar{\boldsymbol{u}}^t, i$
$h_i(\bar{\boldsymbol{u}}^t)$

Dual scenario subproblem



Slave s_1 Evaluating f_0^{μ}, f_0 …

$\sum_{i=1}^{N} \boldsymbol{x}_i^k, \bar{\boldsymbol{u}}^{\text{best}}, \sum_{i=1}^{N} \boldsymbol{x}_i^{\ell},$
$\sum_{i=1}^{N} f_i(\boldsymbol{x}_i^{\ell})$

$\nabla f_0^{\mu}(\boldsymbol{x}^k), f(\boldsymbol{x}^{\ell})$

Slave s_3 Projecting onto $\mathcal{V}$ …

$\tilde{\boldsymbol{v}}$

$\bar{\boldsymbol{u}}$

Master

$\boldsymbol{x}^k_{\mathrm{j}(k)}$

$\mathrm{g}(\mathrm{j}(k), \boldsymbol{x}^k_{\mathrm{j}(k)}),$
$\mathrm{f}_{\mathrm{j}(k)}(\boldsymbol{x}^k_{\mathrm{j}(k)})$

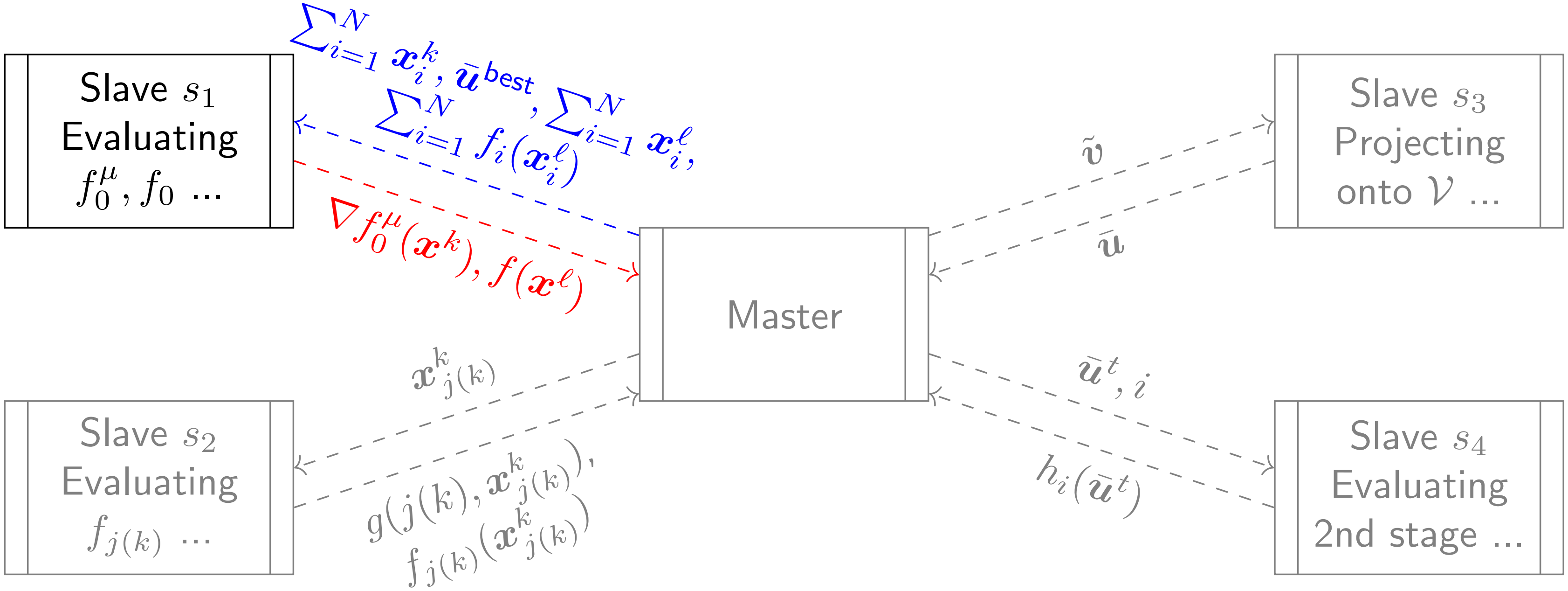
Slave s_2 Evaluating $f_{j(k)}$ …

$\bar{\boldsymbol{u}}^t, i$

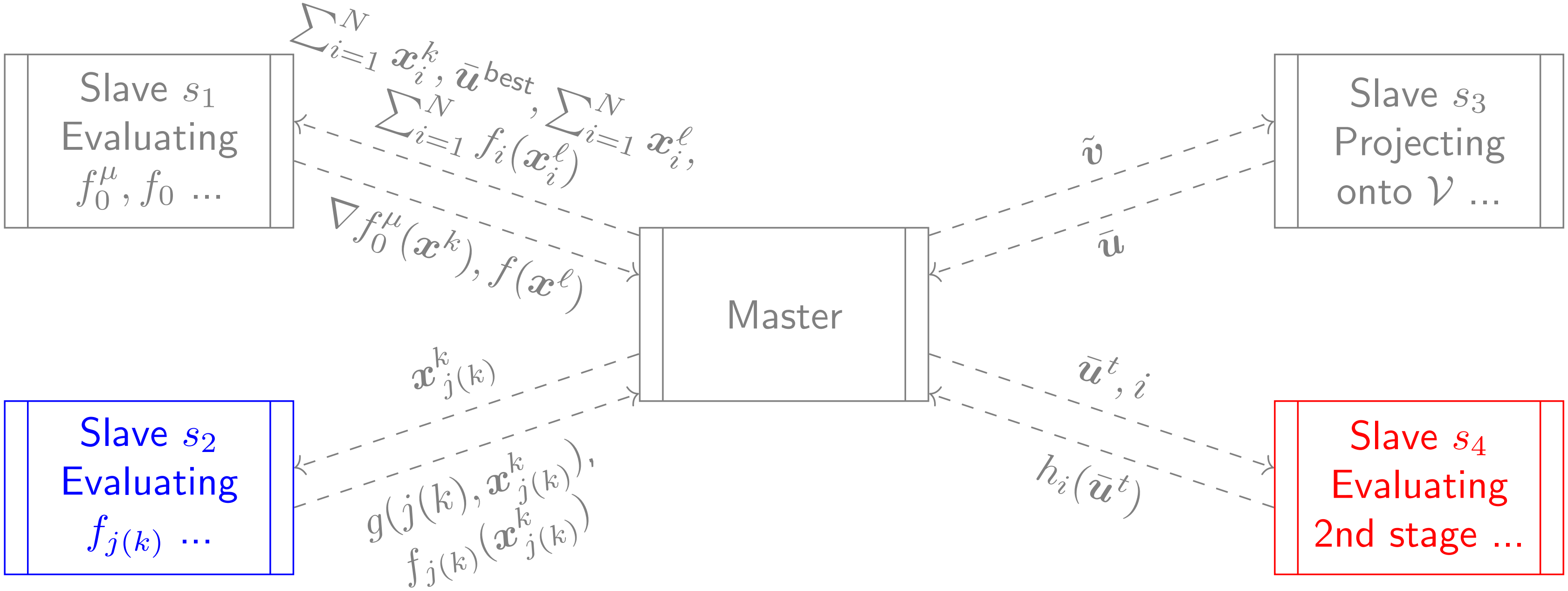
$h_i(\bar{\boldsymbol{u}}^t)$

Slave s_4 Evaluating 2nd stage …

Non-scenario dual subproblems


Slave s_1
Evaluating
f_0^{μ}, f_0 ...
$\sum_{i=1}^{N} \boldsymbol{x}_i^k, \bar{\boldsymbol{u}}^{\text{best}}, \sum_{i=1}^{N} \boldsymbol{x}_i^{\ell},$
$\sum_{i=1}^{N} f_i(\boldsymbol{x}_i^{\ell})$
$\nabla f_0^{\mu}(\boldsymbol{x}^k), f(\boldsymbol{x}^{\ell})$
Slave s_3
Projecting
onto $\mathcal{V}$ ...
$\tilde{\boldsymbol{v}}$
$\bar{\boldsymbol{u}}$
Master
$\boldsymbol{x}_{j(k)}^{k}$
Slave s_2
Evaluating
$f_{j(k)}$ ...
$g(j(k), \boldsymbol{x}_{j(k)}^{k}),$
$f_{j(k)}(\boldsymbol{x}_{j(k)}^{k})$
$\bar{\boldsymbol{u}}^t, i$
$h_i(\bar{\boldsymbol{u}}^t)$
Slave s_4
Evaluating
2nd stage ...

Intensively parallel tasks



Slave s_1
Evaluating
f_0^{μ}, f_0 …

$\sum_{i=1}^{N} \boldsymbol{x}_i^k, \bar{\boldsymbol{u}}^{\text{best}}, \sum_{i=1}^{N} \boldsymbol{x}_i^{\ell},$

$\sum_{i=1}^{N} f_i(\boldsymbol{x}_i^{\ell})$

$\nabla f_0^{\mu}(\boldsymbol{x}^k), f(\boldsymbol{x}^{\ell})$

Master

$\tilde{\boldsymbol{v}}$

$\bar{\boldsymbol{u}}$

Slave s_3
Projecting
onto $\mathcal{V}$ …

$\boldsymbol{x}^k_{j(k)}$

$g(j(k), \boldsymbol{x}^k_{j(k)}),$
$f_{j(k)}(\boldsymbol{x}^k_{j(k)})$

Slave s_2
Evaluating
$f_{j(k)}$ …

$\bar{\boldsymbol{u}}^t, i$

$h_i(\bar{\boldsymbol{u}}^t)$

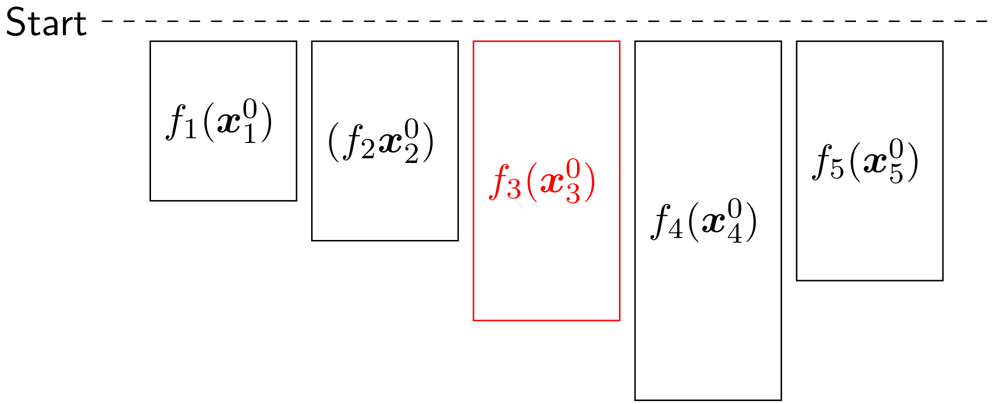
Slave s_4
Evaluating
2nd stage …

Master operations

1. Wait for next result from any Slave
2. Receive result from Slave s and pos-process it
3. Check bounds: if $UB - LB < \epsilon$ terminate
4. Decide next task
5. If task is *dual scenario*:
   (a) Select j at random from $\{1, \ldots, N\}$ and update $\boldsymbol{x}_j$
   (b) Send *new* $\boldsymbol{x}_j$ to Slave s

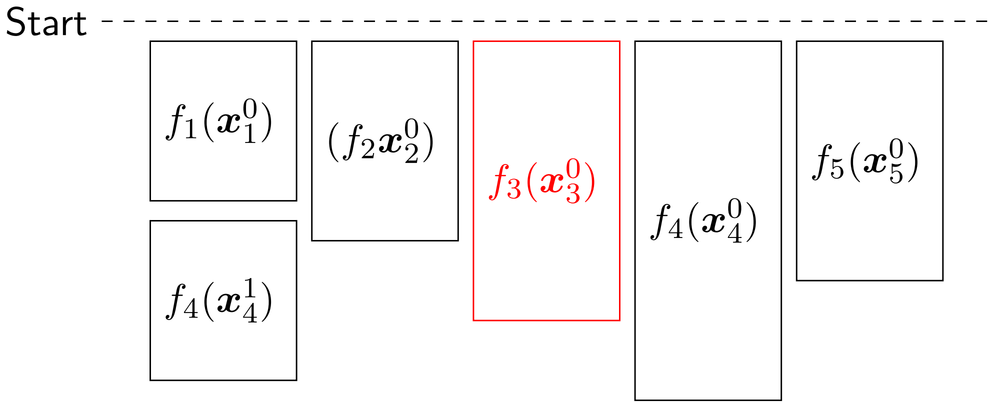
   Else, pre-process task accordingly and send it to Slave s
6. Return to 1.

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$

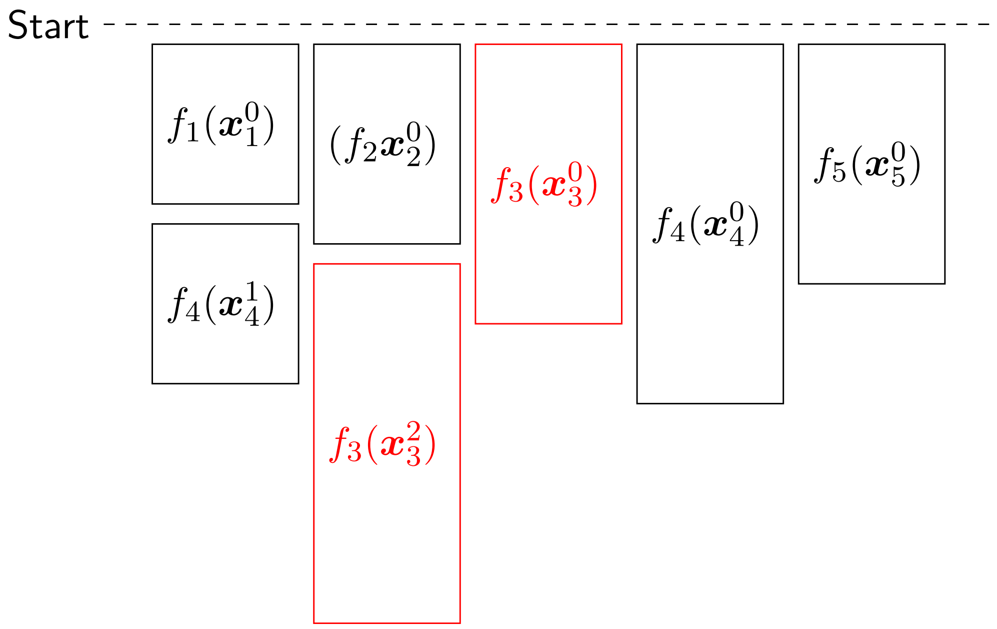
k	$j(k)$

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$

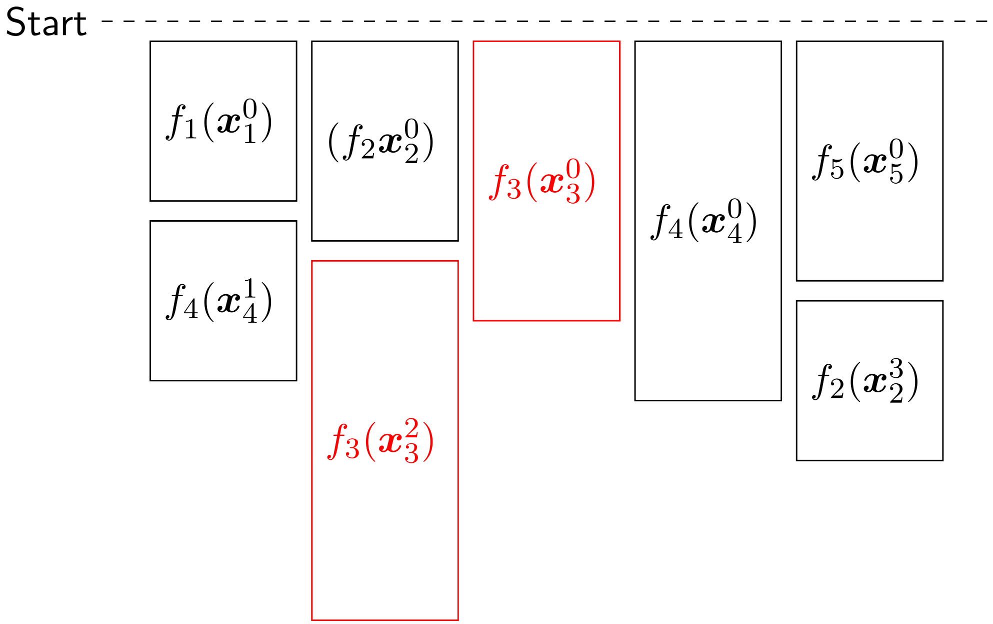
k	$j(k)$
1	4

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_3(\boldsymbol{x}_3^2)$

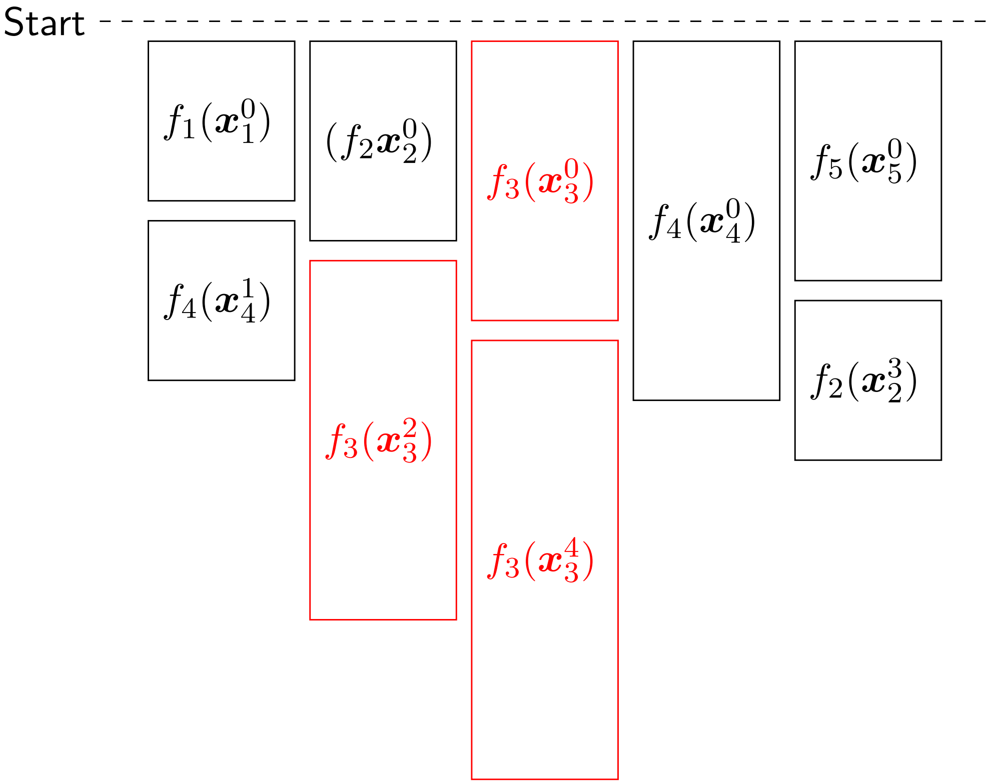
k	$j(k)$
1	4
2	3

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_3(\boldsymbol{x}_3^2)$
$f_2(\boldsymbol{x}_2^3)$

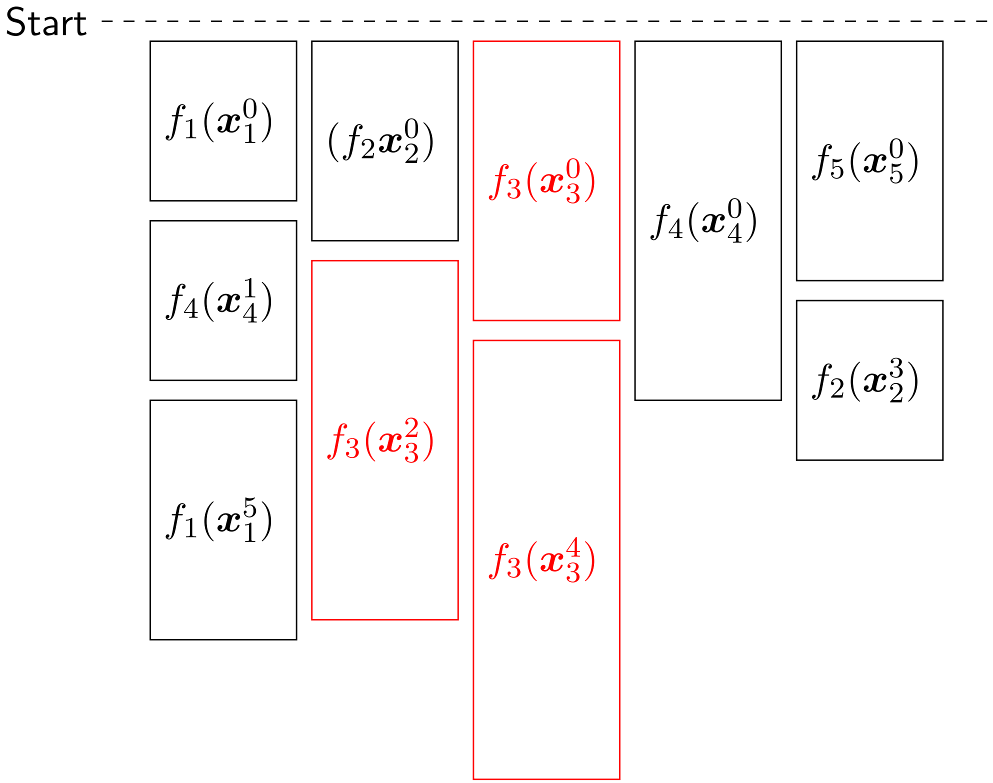
k	$j(k)$
1	4
2	3
3	2

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$f_4(\boldsymbol{x}_4^1)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^2)$
$f_3(\boldsymbol{x}_3^0)$
$f_3(\boldsymbol{x}_3^4)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_2(\boldsymbol{x}_2^3)$

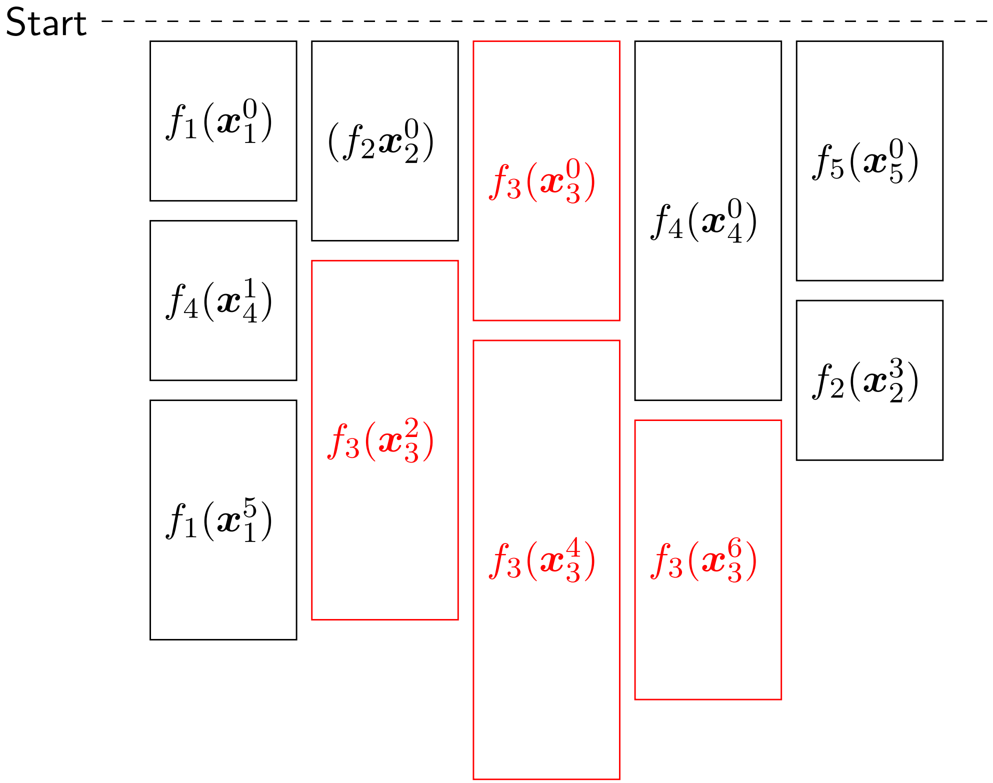
k	$j(k)$
1	4
2	3
3	2
4	3

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_3(\boldsymbol{x}_3^2)$
$f_2(\boldsymbol{x}_2^3)$
$f_1(\boldsymbol{x}_1^5)$
$f_3(\boldsymbol{x}_3^4)$

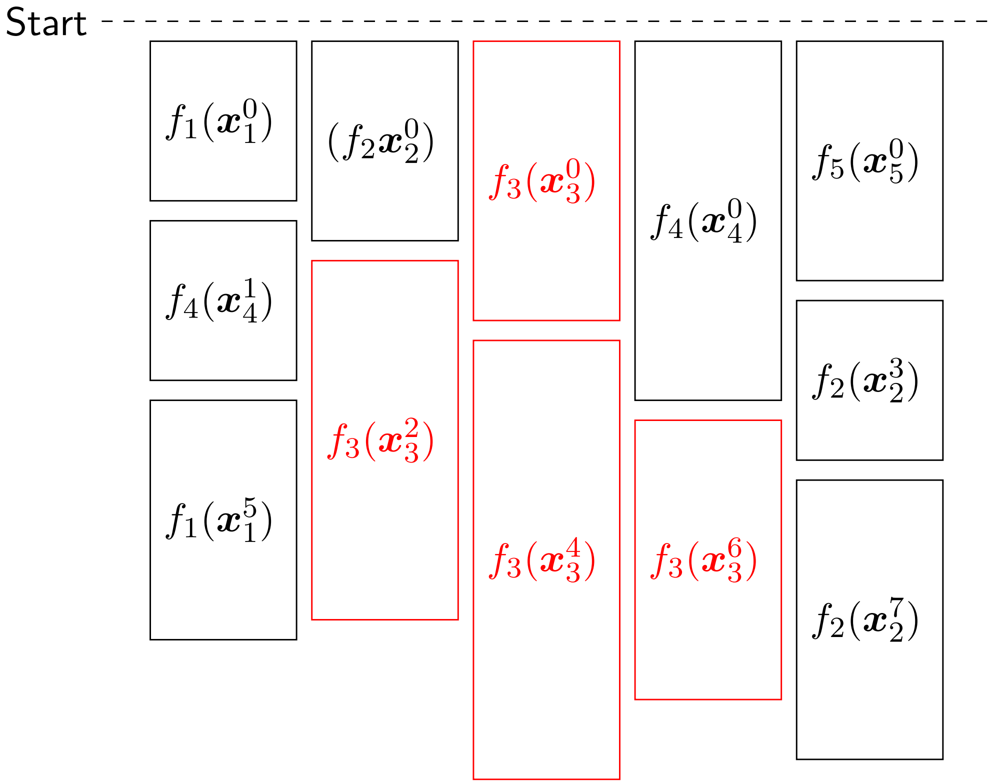
k	$j(k)$
1	4
2	3
3	2
4	3
5	1

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_3(\boldsymbol{x}_3^2)$
$f_2(\boldsymbol{x}_2^3)$
$f_1(\boldsymbol{x}_1^5)$
$f_3(\boldsymbol{x}_3^4)$
$f_3(\boldsymbol{x}_3^6)$

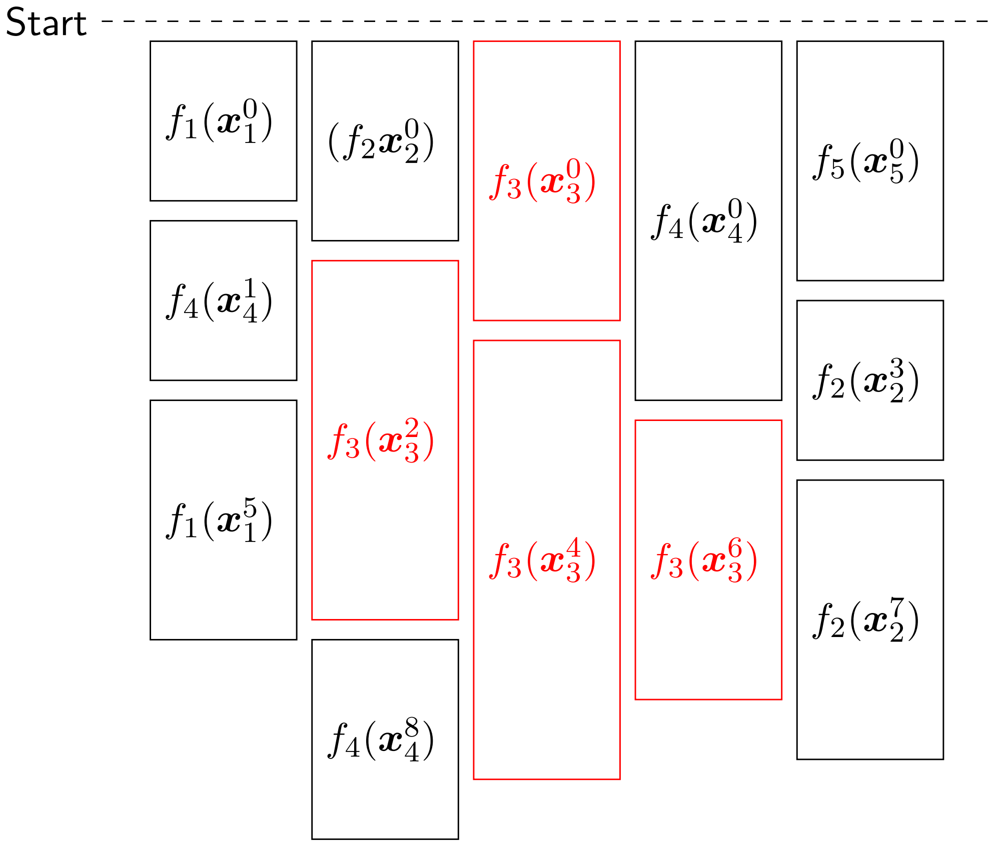
k	$j(k)$
1	4
2	3
3	2
4	3
5	1
6	3

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_1(\boldsymbol{x}_1^5)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^2)$
$f_3(\boldsymbol{x}_3^0)$
$f_3(\boldsymbol{x}_3^4)$
$f_4(\boldsymbol{x}_4^0)$
$f_3(\boldsymbol{x}_3^6)$
$f_5(\boldsymbol{x}_5^0)$
$f_2(\boldsymbol{x}_2^3)$
$f_2(\boldsymbol{x}_2^7)$

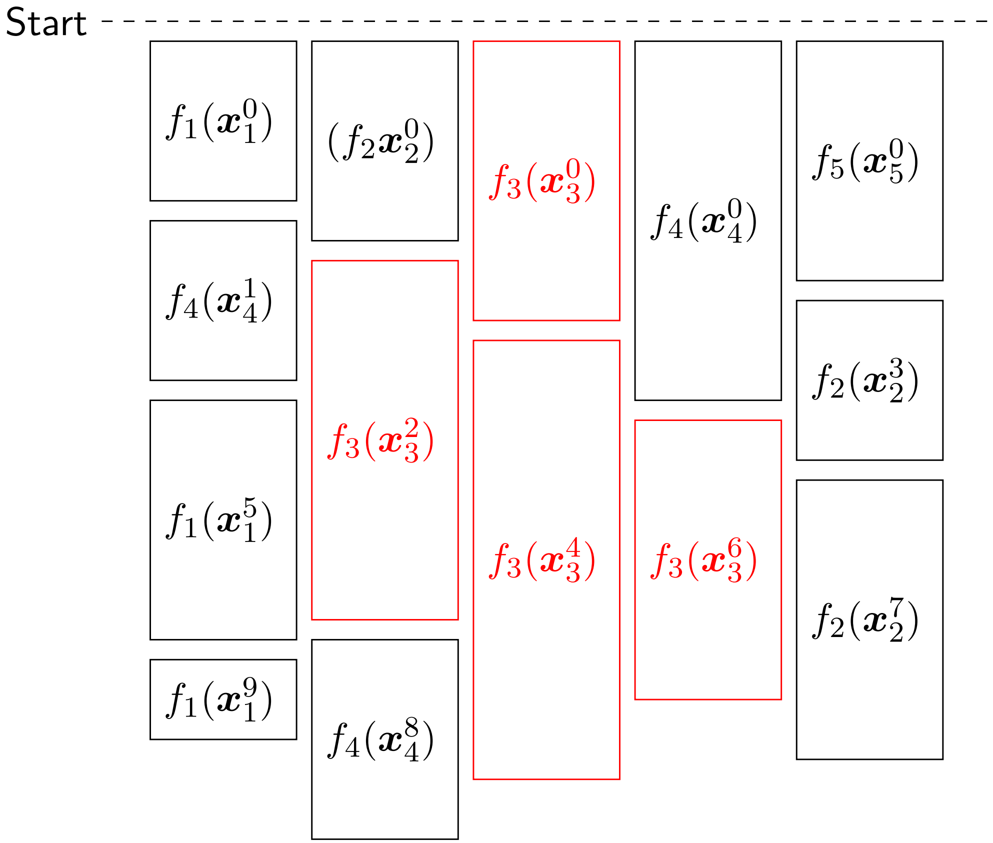
k	$j(k)$
1	4
2	3
3	2
4	3
5	1
6	3
7	2

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_3(\boldsymbol{x}_3^2)$
$f_2(\boldsymbol{x}_2^3)$
$f_1(\boldsymbol{x}_1^5)$
$f_3(\boldsymbol{x}_3^4)$
$f_3(\boldsymbol{x}_3^6)$
$f_2(\boldsymbol{x}_2^7)$
$f_4(\boldsymbol{x}_4^8)$

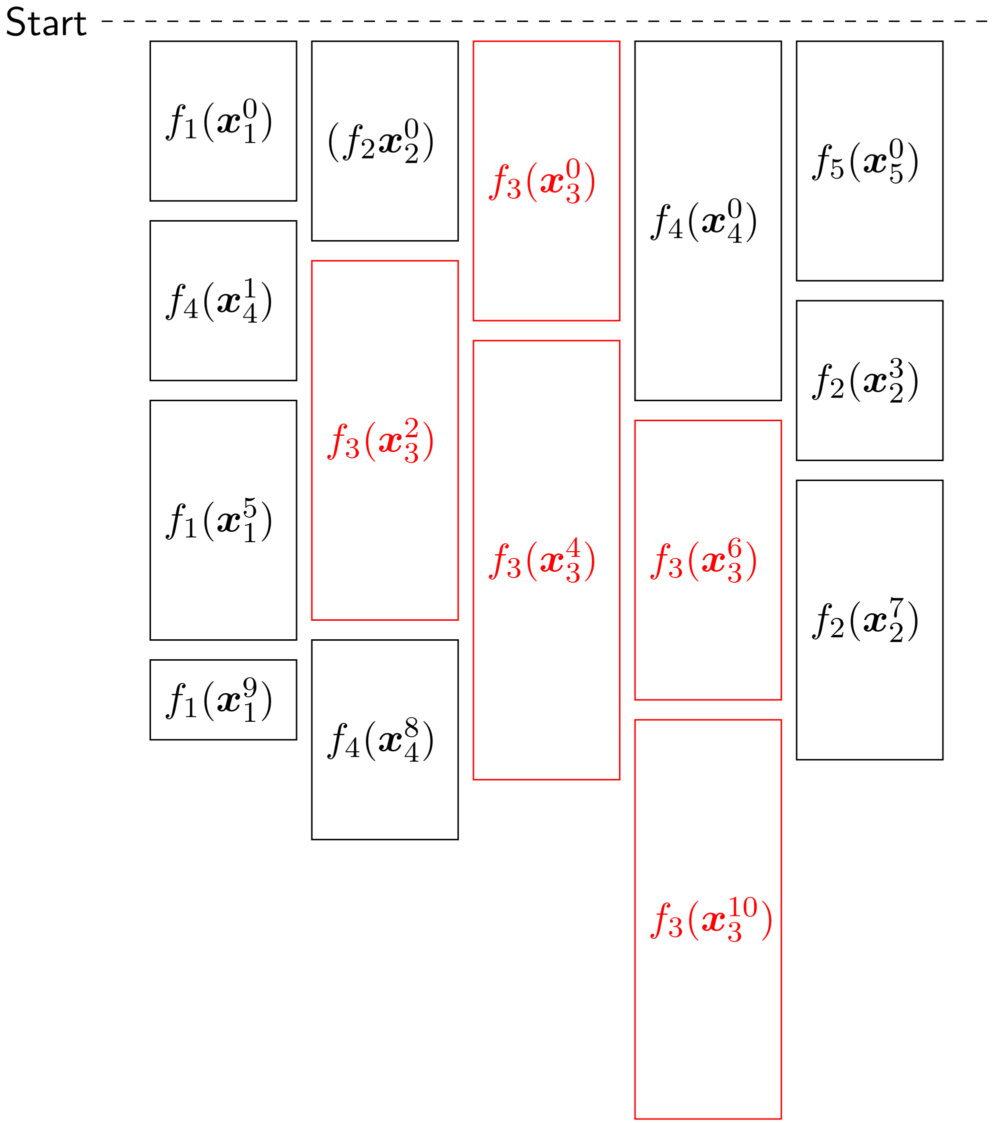
k	$j(k)$
1	4
2	3
3	2
4	3
5	1
6	3
7	2
8	4

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_1(\boldsymbol{x}_1^5)$
$f_1(\boldsymbol{x}_1^9)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^2)$
$f_4(\boldsymbol{x}_4^8)$
$f_3(\boldsymbol{x}_3^0)$
$f_3(\boldsymbol{x}_3^4)$
$f_4(\boldsymbol{x}_4^0)$
$f_3(\boldsymbol{x}_3^6)$
$f_5(\boldsymbol{x}_5^0)$
$f_2(\boldsymbol{x}_2^3)$
$f_2(\boldsymbol{x}_2^7)$

k	$j(k)$
1	4
2	3
3	2
4	3
5	1
6	3
7	2
8	4
9	1

Built-in load balancing for subgradient computation


Start
$f_1(\boldsymbol{x}_1^0)$
$(f_2\boldsymbol{x}_2^0)$
$f_3(\boldsymbol{x}_3^0)$
$f_4(\boldsymbol{x}_4^0)$
$f_5(\boldsymbol{x}_5^0)$
$f_4(\boldsymbol{x}_4^1)$
$f_3(\boldsymbol{x}_3^2)$
$f_2(\boldsymbol{x}_2^3)$
$f_1(\boldsymbol{x}_1^5)$
$f_3(\boldsymbol{x}_3^4)$
$f_3(\boldsymbol{x}_3^6)$
$f_2(\boldsymbol{x}_2^7)$
$f_1(\boldsymbol{x}_1^9)$
$f_4(\boldsymbol{x}_4^8)$
$f_3(\boldsymbol{x}_3^{10})$

k	$j(k)$
1	4
2	3
3	2
4	3
5	1
6	3
7	2
8	4
9	1
10	3

Numerical results and conclusions

Numerical results

□ The asynchronous algorithm is implemented in C using the Xpress C API and using MPI to handle communications between subprocesses.

□ Intances read from SMPS files with explicit periods, preserving *lazy constraints*.

□ Subproblems were solved to 1% optimality.

□ Numerical experiments ran on the Cab cluster, hosted at the Lawrence Livermore National Laboratory.

□ We ran experiments using the WECC system (Papavasiliou et al., 2015) and the CWE system (Aravena and Papavasiliou, 2017)

□ 8 day types per instance: 4 seasons, one weekday and one weekend day per season

Problem sizes

Stochastic unit commitment instances in the literature.

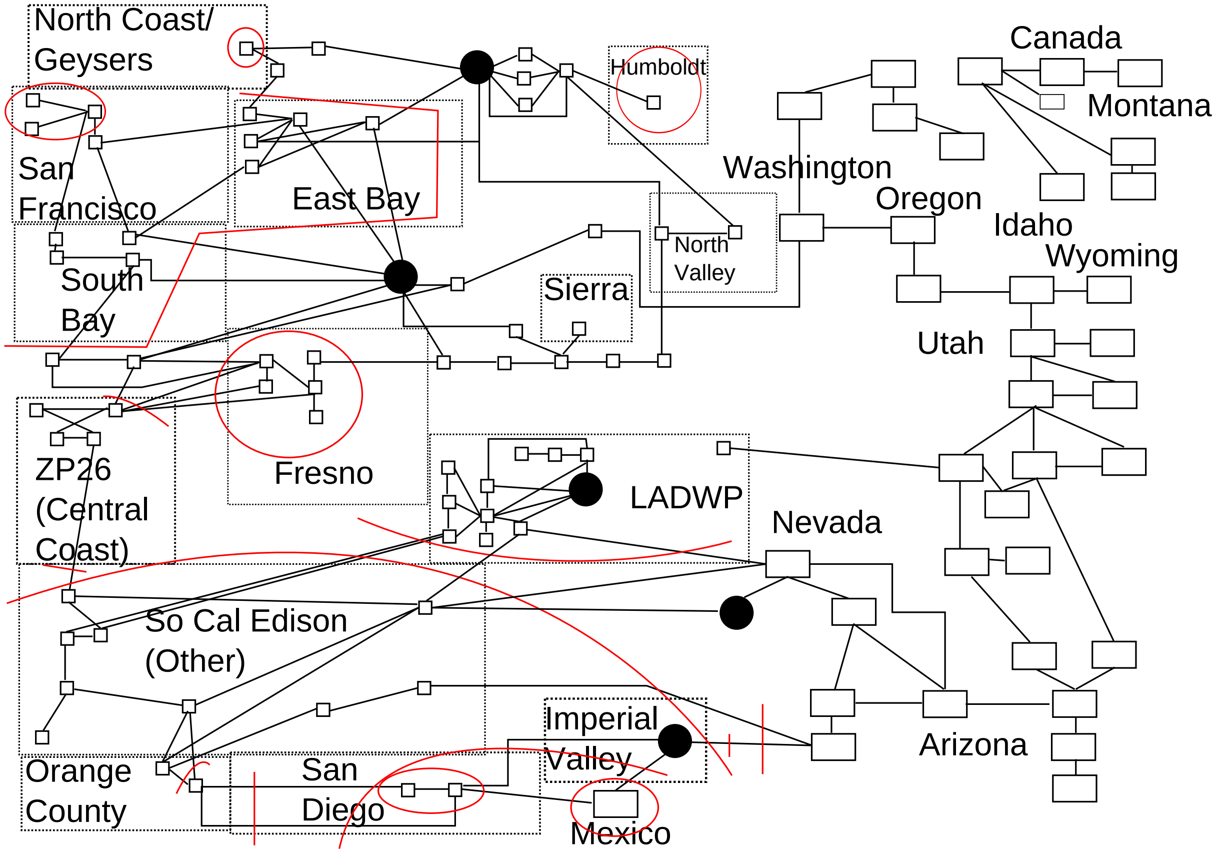
Instance	Rows	Columns	Non-zeros	Integers	Subproblem solution time [s], avg.	(max.)
WECC[1]	69 447	28 943	240 724	4 080	9.4	(25.7)
WECC[2]	34 441	23 090	139 394	3 074	8.3	(67.9)
EDF[3]	812 906	73 562	–	26 122	–	–
CWE[4]	609 589	390 075	1 941 270	9 753	3 383.2	(7 851.8)

Largest instances solved in the present study.

Instance	Scenarios	Rows	Columns	Integers
WECC-182[2]	1000	36 267 000	23 091 826	3 074 000
CWE[4]	120	74 755 320	46 822 372	1 170 360

[1](Cheung et al., 2015), [2](Papavasiliou et al., 2015), [3](van Ackooij and Malick, 2016), [4](Aravena and Papavasiliou, 2017)

Western Electricity Coordinating Council

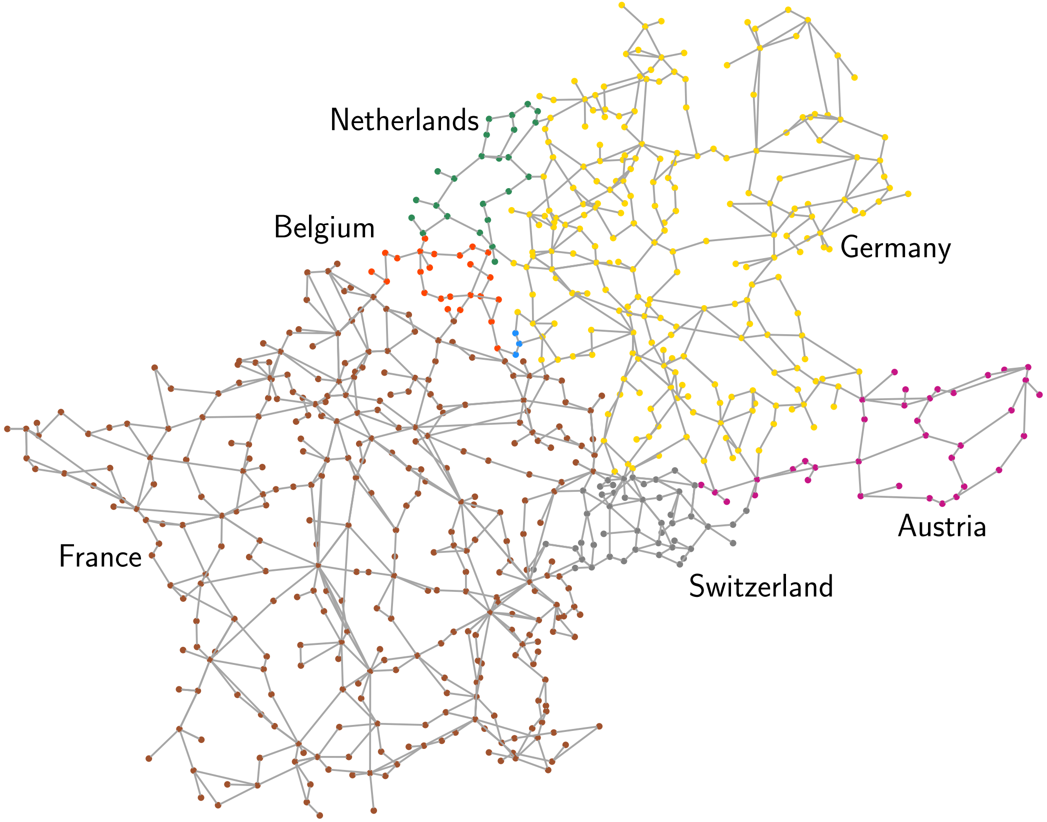


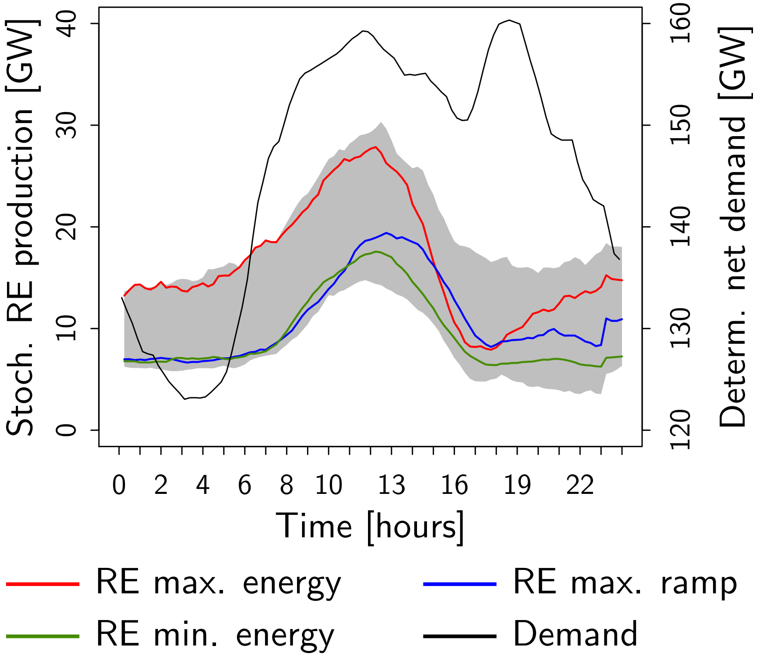
130 thermal generators, 182 nodes, 319 lines, hourly resolution, 24 hour horizon, generation and transmission contingencies, multi-area renewable production

WECC: Solution times

N	Step size	*Recovery Type*	# Cores	*Dual Share*	Solution time [s], avg. (max.)			
					2% optimality		1% optimality	
10	Dim. 1/k	FIFO	16	0.5	228.1	(792.9)	–	
	Dim. 1/k	RND	16	0.5	229.4	(856.1)	–	
	Dim. 1/k	LIFO	16	0.5	200.6	(739.7)	–	
	Dim. 1/k	IS	16	0.5	178.0	(638.0)	–	
	Polyak	FIFO	16	0.5	148.2	(469.3)	424.4	(1 361.1)
	Polyak	RND	16	0.5	117.8	(392.6)	–	
	Polyak	LIFO	16	0.5	131.2	(446.2)	384.0	(1 326.9)
	Polyak	**IS**	**16**	**0.5**	**118.4**	**(441.4)**	**364.7**	**(1 291.5)**
100	Polyak	FIFO	160	0.5	267.6	(325.1)	–	
	Polyak	RND	160	0.5	113.2	(345.6)	534.2	(1 134.1)
	Polyak	LIFO	160	0.5	99.4	(268.3)	508.9	(1 152.4)
	Polyak	**IS**	**160**	**0.5**	**95.5**	**(289.8)**	**517.9**	**(1 126.1)**
1000	Polyak	LIFO	256	0.75	723.9	(2 155.2)	–	
	Polyak	**IS**	**256**	**0.75**	**411.7**	**(1 354.5)**	**2 535.0**	**(6 427.0)**

Central Western European system





656 thermal generators, 679 nodes, 1073 lines, quarterly resolution, 24 hour horizon, multi-area renewable production

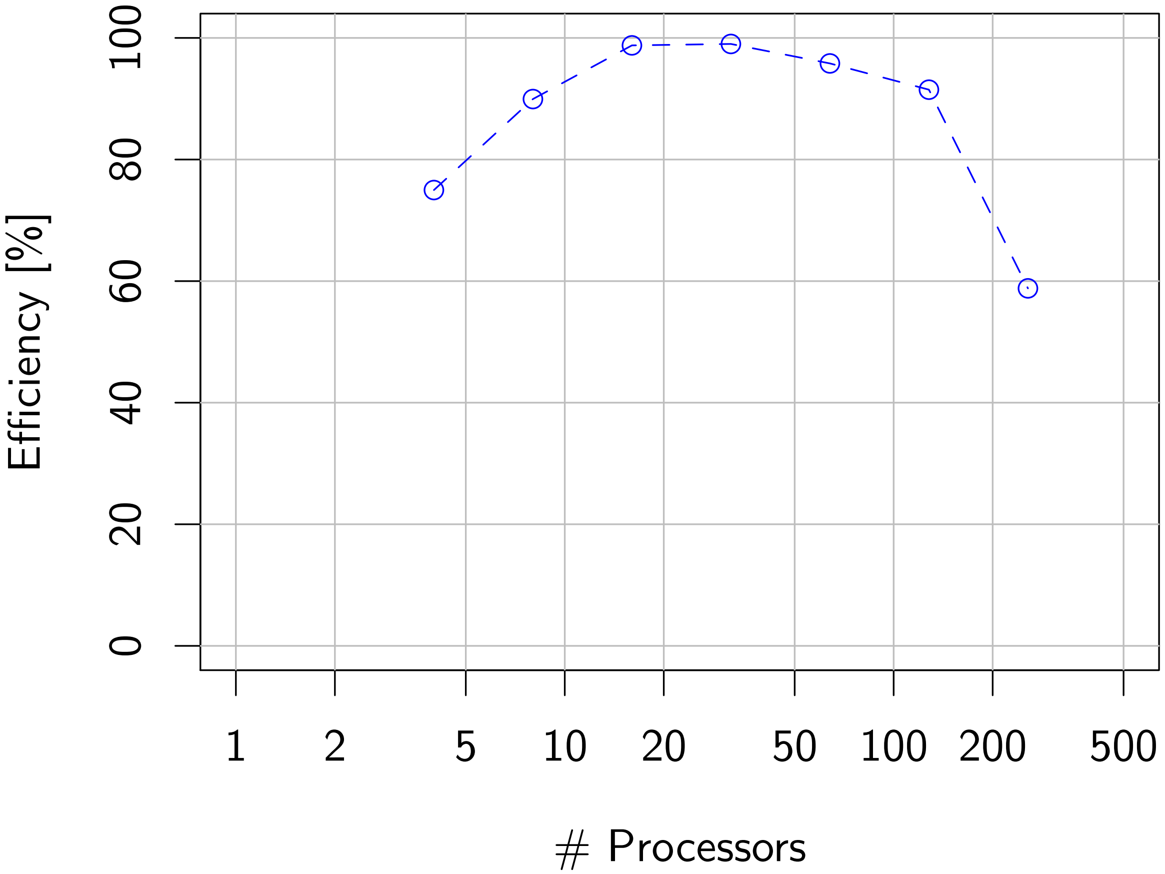
CWE: Solution times

Solution time statistics for WECC instances, over 8 representative day types. All instances use the Polyak stepsize to perform dual updates, the IS primal recovery heuristic and a *DualShare* of 0.75.

N	# Cores	Solution time [s], avg. (max.)			
		2% optimality		1% optimality	
30	96	2 580.3	(5 908.2)	3 806.2	(9 279.1)
60	192	2 563.7	(5 593.3)	3 774.2	(8 323.4)
120	384	2 696.5	(5 973.0)	3 876.2	(7 952.6)

Parallel efficiency

Parallel efficiency plot of the asynchronous algorithm. Plot drawn using WECC, spring weekdays, 100 scenario instance, solved using *Dual Share* 0.75, Polyak stepsize and IS primal recovery.



Synchronous algorithm idle times

Estimated idle time of processors when solving SUC using a parallel synchronous algorithm. Average and maximum over 8 day types.

System	N	# Cores	*Dual Share*	Synchronous idle time [%], avg. (max.)					
				Same conditions		Half # cores		$1+2N$ processes	
WECC	10	16	0.5	48.0	(53.2)	28.3	(34.5)	54.2	(59.2)
	100	160	0.5	70.6	(**80.4**)	52.9	(**65.0**)	74.3	(83.8)
	1000	256	0.75	37.6	(57.0)	21.1	(36.9)	85.4	(**92.4**)
CWE	30	96	0.75	36.4	(47.0)	27.1	(33.9)	46.5	(53.9)
	60	192	0.75	43.6	(**62.2**)	33.4	(**47.3**)	53.3	(**67.2**)
	120	384	0.75	46.9	(61.2)	36.3	(46.6)	54.7	(65.4)

Conclusions

□ Randomized block-coordinate descent converges without the need for line search

□ Synchronous algorithms can lead to idle times of up to 80.2% of the total wall time → if you want to **go parallel**, then **go asynchronous**

□ The asynchronous algorithm allows us to **solve industrial-scale instances** operationally acceptable time frames

□ Future extensions of the present work will focus on:

- Extensions to multi-stage stochastic unit commitment
- Integrated optimization of generation, transmission and distribution systems (Caramanis et al., 2016)

Thank you

Contact:

Ignacio Aravena, ignacio.aravena@uclouvain.be
http://sites.google.com/site/iaravenasolis/

Anthony Papavasiliou, anthony.papavasiliou@uclouvain.be
http://perso.uclouvain.be/anthony.papavasiliou/

Appendix

Assumptions

1. **Subgradient boundedness**

   There exist C and D such that

$$\sup_{\substack{j\in\{1,\cdots,N\}\\ x,y\in X}} \left\|I_j \nabla f_0^{\mu}\big(\boldsymbol{x}\big) + g\big(j, \boldsymbol{y}_j\big)\right\|_2 \leq C, \qquad \sup_{\substack{j\in\{1,\cdots,N\}\\ x\in X}} \left\|g\big(j, \boldsymbol{x}_j\big)\right\|_2 \leq D.$$

2. **Delay boundedness**

   There exists L such that $l(k) \leq L$ and $\ell(i,k) \leq L$, $\forall i, k$

3. **Diminishing-bounded stepsize**

   λ_k is independent from $j(k)$ and there exist positive constants $\check{G}$ and $\hat{G}$, such that

$$\check{G}\gamma_k \leq \lambda_k \leq \hat{G}\gamma_k, \ \gamma_k = \frac{1}{(1+rk)^q} \ \ \forall k, \quad \sum_{k=0}^{\infty} \gamma_k = \infty, \ \ \sum_{k=0}^{\infty} \gamma_k^2 < \infty$$

Incremental method

□ Each iteration uses gradient information of a part of the objective function (Nedić et al., 2001), (Gürbüzbalaban et al., 2015)

$$\min_{x \in \mathbb{R}^N} \sum_i f_i(\boldsymbol{x})$$

1. Let $k := 0, \boldsymbol{x}^k := \bar{\boldsymbol{x}}^0$
2. Select component $j(k)$ (cyclically, at random)
3. Compute $\boldsymbol{g}^k \in \partial f_{j(k)}(\boldsymbol{x}^k)$
4. Let $\boldsymbol{x}^{k+1} := \boldsymbol{x}^k - \lambda_k \, \boldsymbol{g}^k$
5. Let $k := k + 1$ and return to 2.

Block-coordinate descent method

□ Each iteration performs a line search on a subset of variables (Tseng and Yun, 2009), (Fercoq and Richtárik, 2013), (Wright, 2015)

$$\min_{x \in \mathbb{R}^N} \; f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$$

1. Let $k := 0, \boldsymbol{x}^k := \bar{\boldsymbol{x}}^0$

2. Select block-coordinate $j(k)$ (cyclically, at random)

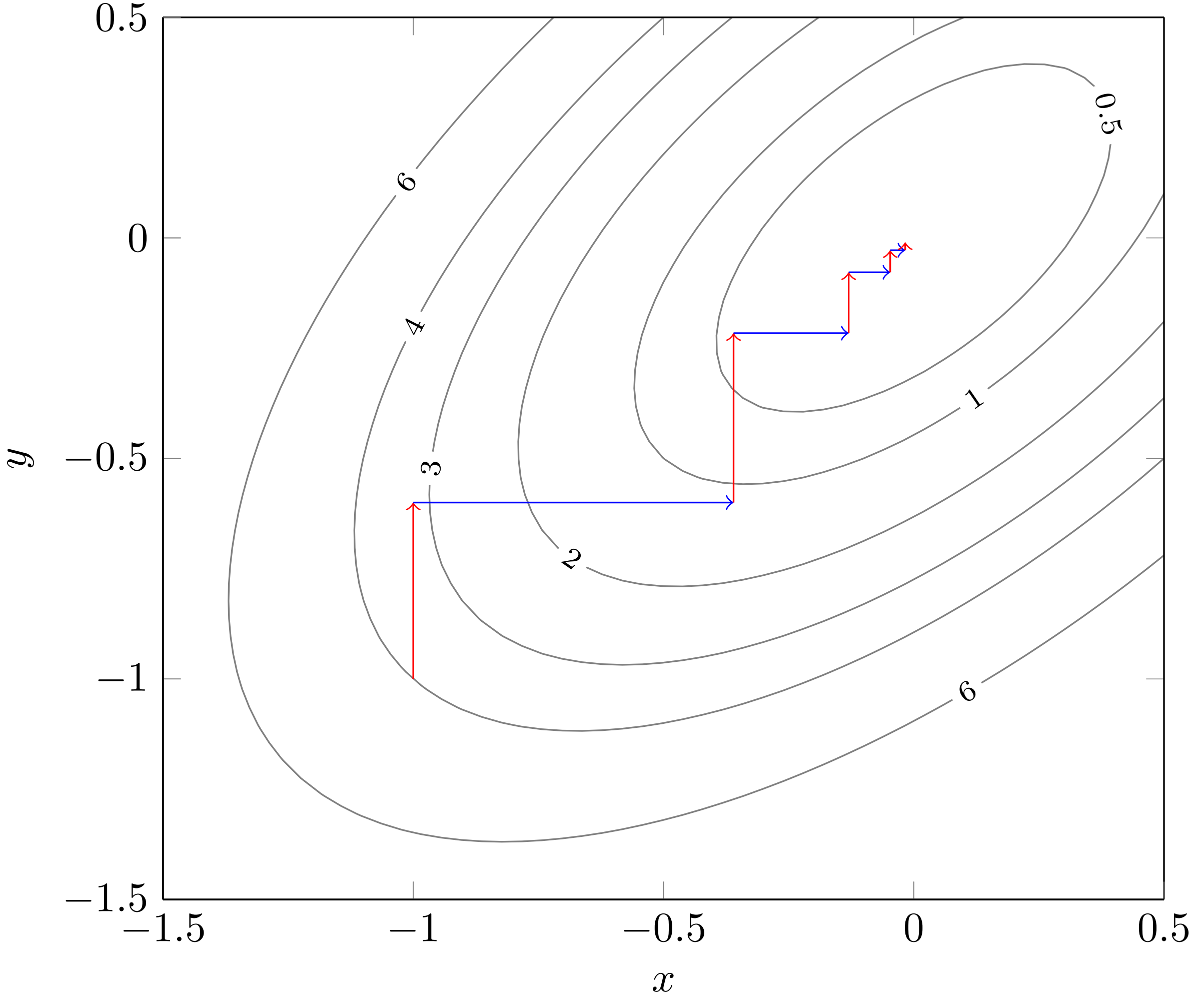
3. Compute $\boldsymbol{t}^k := \arg\min_t f(\boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_{j(k)-1}^k, \boldsymbol{t}, \boldsymbol{x}_{j(k)+1}^k, \ldots, \boldsymbol{x}_n^k)$

4. Let $\boldsymbol{x}_j^{k+1}(k) := \boldsymbol{t}^k \,,\; \boldsymbol{x}_i^{k+1} := \boldsymbol{x}_i^k \;\; \forall i \neq j(k)$

5. Let $k := k+1$ and return to 2.

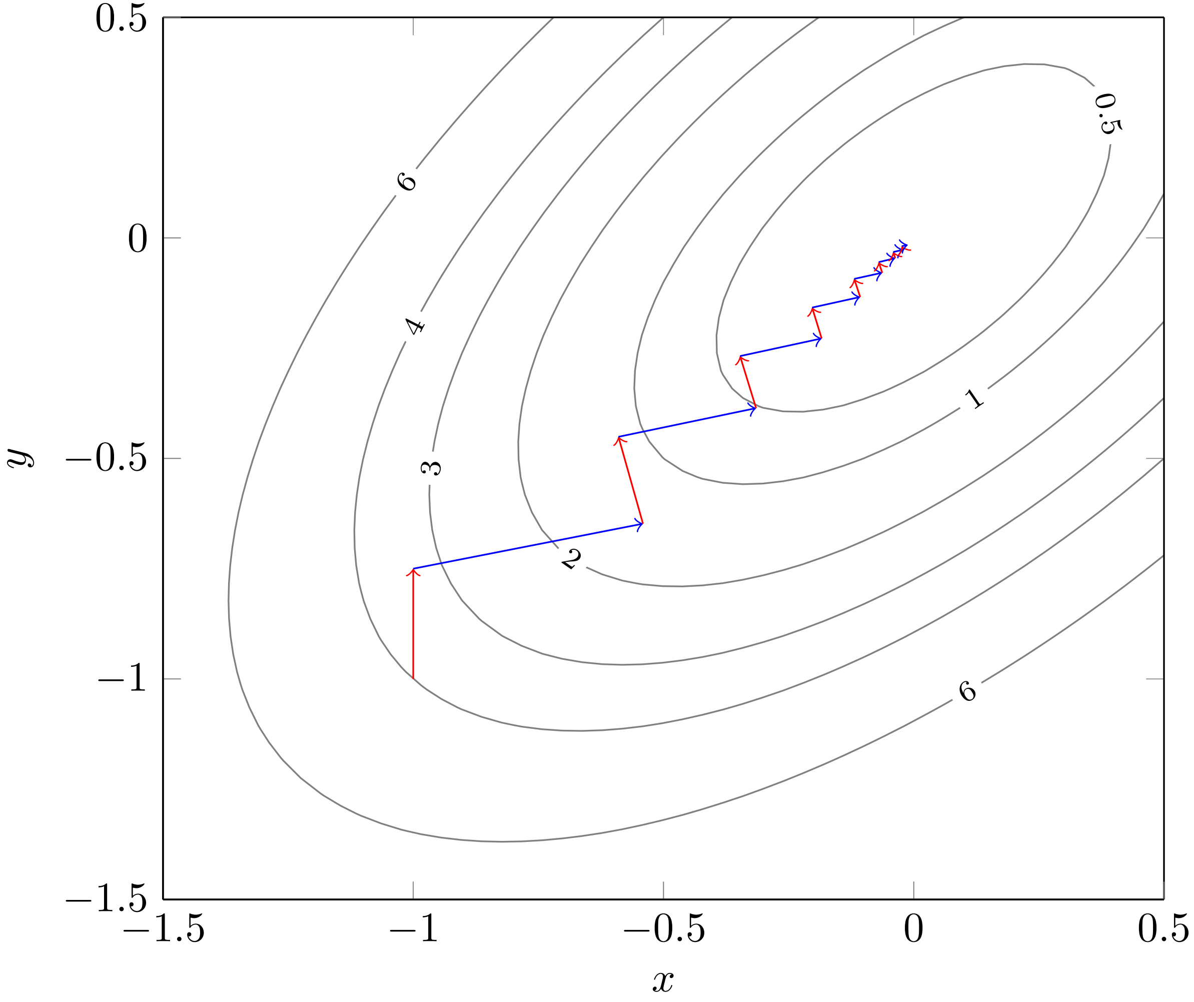
Block-coordinate descent method

$$f(x, y) = 5x^2 - 6xy + 5y^2$$



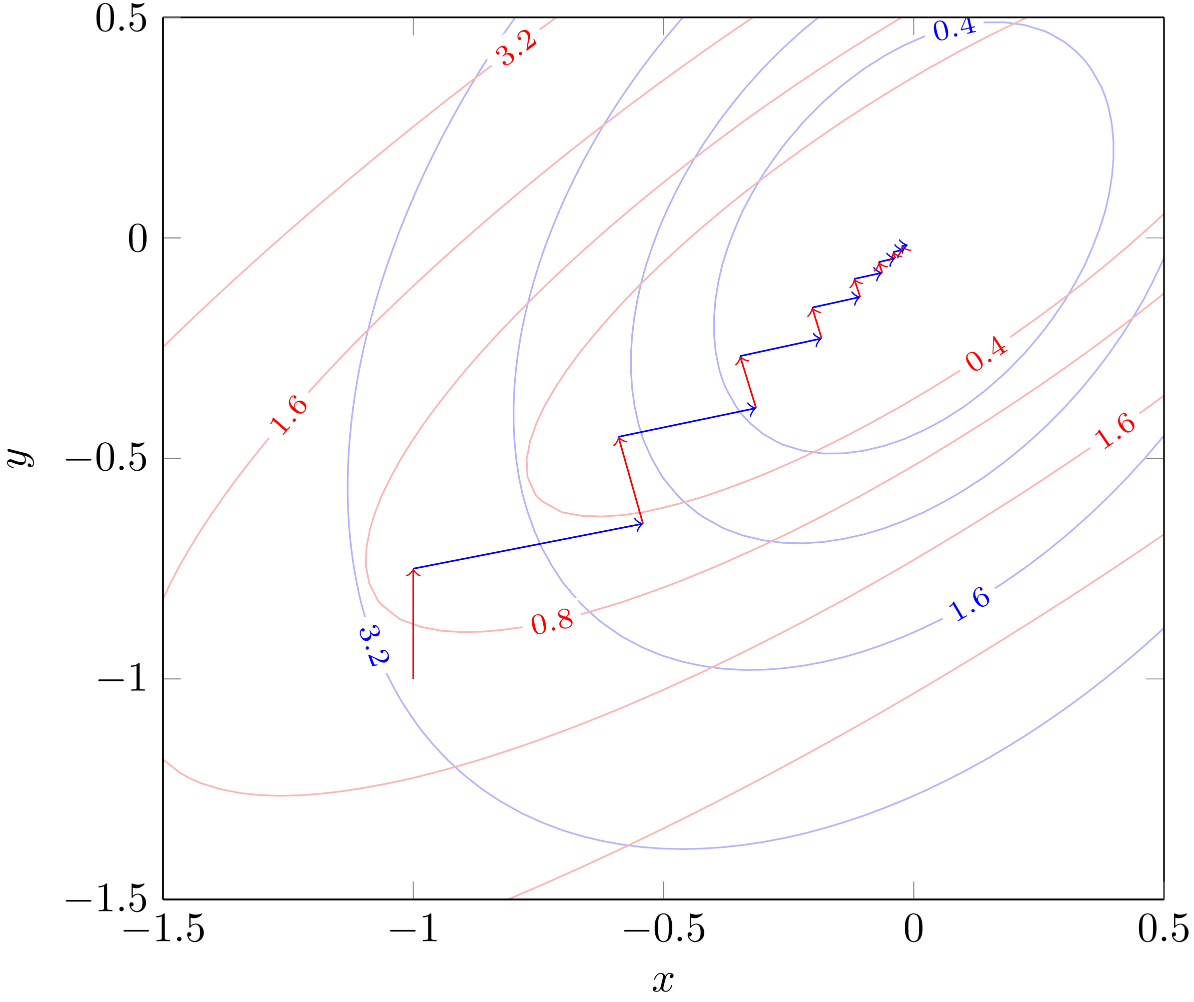
Incremental method

$$f(x,y) = 3x^2 - 2xy + 2y^2 + 2x^2 - 4xy + 3y^2$$



Incremental method

$$f(x,y) = 3x^2 - 2xy + 2y^2 + 2x^2 - 4xy + 3y^2$$



Asynchronous BCD vs asynchronous incremental method

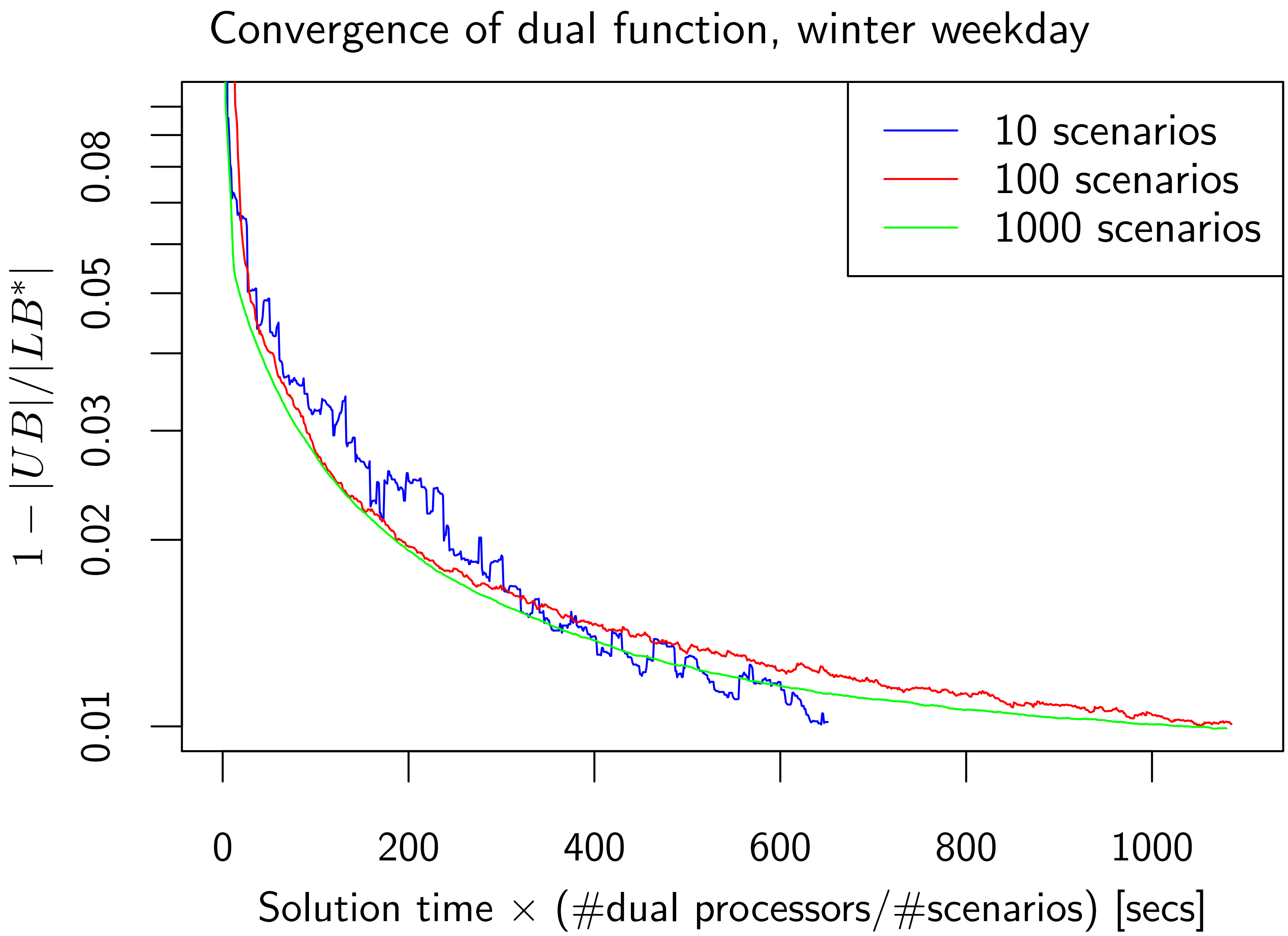
□ The asynchronous incremental method (Nedić, 2001) could be readily applied to minimize f. We lean towards coordinate descent because

- As $\boldsymbol{x}$ approaches an optimal value, the gradient of f_0^μ will tend to point in the opposite direction to the subgradient of f_i. This makes the incremental method susceptible to oscillations.

- Every time the asynchronous incremental method updates $\boldsymbol{x}$ using the gradient of f_0^μ, it will introduce an **additional delay on all subgradients currently being computed**.

  Each block-coordinate update causes an **additional delay only on block** $j(k)$, if $g(j(k), \boldsymbol{x}_{j(k)}^{k-l'(j(k),k)})$ is being computed.

WECC: Dual algorithm progress



Solution time sensitivity to primal/dual allocation of resources

Variation of solution time with the *Dual Share*. Results in the table correspond to WECC, spring weekdays, 100 scenario instance, solved using 8 nodes (96 cores) and a Polyak stepsize.

Primal	*Dual Share*		Solution time [s], avg. (max.)			
Recovery	$k = 0$	$k = 200N$	2% optimality		1% optimality	
IS	0.1	0.1	150.5	(168.7)	1731.9	(1821.0)
	0.25	0.25	78.9	(82.8)	785.7	(807.8)
	0.5	0.5	52.5	(55.6)	441.4	(467.6)
	0.75	0.75	67.2	(78.0)	307.4	(333.3)
	0.9	0.9	58.1	(72.3)	291.0	(294.2)
LIFO	0.5	0.5	97.5	(120.0)	529.9	(598.3)
	0.75	0.75	92.3	(104.5)	479.9	(624.1)
IS	0.75	0.25	50.9	(59.0)	313.9	(334.5)
	0.9	0.1	66.8	(77.5)	280.3	(320.2)