Convex Optimization

(EE227A: UC Berkeley)

Lecture 21
(BCD – II, Parallel algorithms)

09 Apr, 2013

Suvrit Sra

Admin

- Use following URL to submit your project material
- https://www.easychair.org/conferences/?conf=ee227a2013
- You'll have to sign up at easychair for submitting
- Make sure **each author** is in the system
- Deadline: **4/12/2013; 5pm**

Admin

- Use following URL to submit your project material
- https://www.easychair.org/conferences/?conf=ee227a2013
- You'll have to sign up at easychair for submitting
- Make sure **each author** is in the system
- Deadline: **4/12/2013; 5pm**
- I'll assign the papers for review via easychair

Admin

- Use following URL to submit your project material
- https://www.easychair.org/conferences/?conf=ee227a2013
- You'll have to sign up at easychair for submitting
- Make sure **each author** is in the system
- Deadline: **4/12/2013; 5pm**
- I'll assign the papers for review via easychair
- You'll have to upload review to easychair

Admin

- Use following URL to submit your project material
- https://www.easychair.org/conferences/?conf=ee227a2013
- You'll have to sign up at easychair for submitting
- Make sure **each author** is in the system
- Deadline: **4/12/2013; 5pm**
- I'll assign the papers for review via easychair
- You'll have to upload review to easychair
- Reviews are **per person**
- **No class on 4/11**
- Again: **project submissions are electronic only!**

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i
- Say block 1 contains coordinates $\{1, 3, 5\}$.
  - We write $x^{(1)} = (x_1, x_3, x_5)$

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i
- Say block 1 contains coordinates $\{1, 3, 5\}$.
  - We write $x^{(1)} = (x_1, x_3, x_5)$
  - Alternatively, $x^{(i)} = x_1 \boldsymbol{e}_1 + x_3 \boldsymbol{e}_3 + x_5 \boldsymbol{e}_5$

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i
- Say block 1 contains coordinates $\{1, 3, 5\}$.
  - We write $x^{(1)} = (x_1, x_3, x_5)$
  - Alternatively, $x^{(i)} = x_1 \boldsymbol{e}_1 + x_3 \boldsymbol{e}_3 + x_5 \boldsymbol{e}_5$
  - Define $E_1 := [\boldsymbol{e}_1, \boldsymbol{e}_3, \boldsymbol{e}_5]$ ($N \times 3$ matrix)

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i
- Say block 1 contains coordinates $\{1, 3, 5\}$.
  - We write $x^{(1)} = (x_1, x_3, x_5)$
  - Alternatively, $x^{(i)} = x_1 \boldsymbol{e}_1 + x_3 \boldsymbol{e}_3 + x_5 \boldsymbol{e}_5$
  - Define $E_1 := [\boldsymbol{e}_1, \boldsymbol{e}_3, \boldsymbol{e}_5]$ ($N \times 3$ matrix)
  - Then, $x^{(1)} = E_1^T \boldsymbol{x}$

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i
- Say block 1 contains coordinates $\{1, 3, 5\}$.
  - We write $x^{(1)} = (x_1, x_3, x_5)$
  - Alternatively, $x^{(i)} = x_1 \boldsymbol{e}_1 + x_3 \boldsymbol{e}_3 + x_5 \boldsymbol{e}_5$
  - Define $E_1 := [\boldsymbol{e}_1, \boldsymbol{e}_3, \boldsymbol{e}_5]$ ($N \times 3$ matrix)
  - Then, $x^{(1)} = E_1^T \boldsymbol{x}$
- More generally, say π is a random perm of $[N] := \{1, 2, \ldots, N\}$

Randomized BCD

- Let $\boldsymbol{x} \in \mathbb{R}^N$
- Any vector $\boldsymbol{x} = \sum_{i=1}^{N} x_i \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is **ith canonical basis vector** (ith column of the identity matrix)
- Decompose $\boldsymbol{x}$ into n blocks of size N_i
- Say block 1 contains coordinates $\{1, 3, 5\}$.
  - We write $x^{(1)} = (x_1, x_3, x_5)$
  - Alternatively, $x^{(i)} = x_1 \boldsymbol{e}_1 + x_3 \boldsymbol{e}_3 + x_5 \boldsymbol{e}_5$
  - Define $E_1 := [\boldsymbol{e}_1, \boldsymbol{e}_3, \boldsymbol{e}_5]$ ($N \times 3$ matrix)
  - Then, $x^{(1)} = E_1^T \boldsymbol{x}$
- More generally, say π is a random perm of $[N] := \{1, 2, \ldots, N\}$
- Let E be the permutation of I induced by π

BCD – Decomposition

- **Decomposition:** $E = [E_1, \ldots, E_n]$ into n **blocks**

BCD – Decomposition

- **Decomposition:** $E = [E_1, \ldots, E_n]$ into n **blocks**
- Corresponding decomposition of x is

$$(\underbrace{E_1^T \boldsymbol{x}}_{N_1+}, \underbrace{E_2^T \boldsymbol{x}}_{N_2+}, \ldots, \underbrace{E_n^T \boldsymbol{x}}_{\cdots + N_n = N}) = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$$

BCD – Decomposition

- **Decomposition:** $E = [E_1, \ldots, E_n]$ into n **blocks**
- Corresponding decomposition of x is

$$(\underbrace{E_1^T \boldsymbol{x}}_{N_1+}, \underbrace{E_2^T \boldsymbol{x}}_{N_2+}, \ldots, \underbrace{E_n^T \boldsymbol{x}}_{\cdots +N_n=N}) = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$$

- **Observation:**

$$E_i^T E_j = \begin{cases} I_{N_i} & i = j \\ 0_{N_i,N_j} & i \neq j. \end{cases}$$

BCD – Decomposition

- **Decomposition:** $E = [E_1, \ldots, E_n]$ into n **blocks**
- Corresponding decomposition of x is

$$(\underbrace{E_1^T \boldsymbol{x}}_{N_1+}, \underbrace{E_2^T \boldsymbol{x}}_{N_2+}, \ldots, \underbrace{E_n^T \boldsymbol{x}}_{\cdots +N_n=N}) = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$$

- **Observation:**

$$E_i^T E_j = \begin{cases} I_{N_i} & i = j \\ 0_{N_i,N_j} & i \neq j. \end{cases}$$

- So the E_is define our partitioning of the coordinates

BCD – Decomposition

- **Decomposition:** $E = [E_1, \ldots, E_n]$ into n **blocks**
- Corresponding decomposition of x is

$$(\underbrace{E_1^T \boldsymbol{x}}_{N_1+}, \underbrace{E_2^T \boldsymbol{x}}_{N_2+}, \ldots, \underbrace{E_n^T \boldsymbol{x}}_{\cdots +N_n=N}) = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$$

- **Observation:**

$$E_i^T E_j = \begin{cases} I_{N_i} & i = j \\ 0_{N_i,N_j} & i \neq j. \end{cases}$$

- So the E_is define our partitioning of the coordinates
- Just fancier notation for a random partition of coordinates
- Now with this notation ...

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

Assume gradient of block i is Lipschitz continuous**

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

Assume gradient of block i is Lipschitz continuous**

$$\|\nabla_i f(\boldsymbol{x} + E_i h) - \nabla_i f(\boldsymbol{x})\|_* \leq L_i \|h\|$$

Block gradient $\nabla_i f(\boldsymbol{x})$ is projection of full grad: $E_i^T \nabla f(\boldsymbol{x})$

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

Assume gradient of block i is Lipschitz continuous**

$$\|\nabla_i f(\boldsymbol{x} + E_i h) - \nabla_i f(\boldsymbol{x})\|_* \leq L_i \|h\|$$

Block gradient $\nabla_i f(\boldsymbol{x})$ is projection of full grad: $E_i^T \nabla f(\boldsymbol{x})$

** — each block can use its own norm

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

Assume gradient of block i is Lipschitz continuous**

$$\|\nabla_i f(\boldsymbol{x} + E_i h) - \nabla_i f(\boldsymbol{x})\|_* \leq L_i \|h\|$$

Block gradient $\nabla_i f(\boldsymbol{x})$ is projection of full grad: $E_i^T \nabla f(\boldsymbol{x})$

** — each block can use its own norm

Block Coordinate "Gradient" Descent

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

Assume gradient of block i is Lipschitz continuous**

$$\|\nabla_i f(\boldsymbol{x} + E_i h) - \nabla_i f(\boldsymbol{x})\|_* \leq L_i \|h\|$$

Block gradient $\nabla_i f(\boldsymbol{x})$ is projection of full grad: $E_i^T \nabla f(\boldsymbol{x})$

** — each block can use its own norm

Block Coordinate "Gradient" Descent

- Using the descent lemma, we have blockwise upper bounds

$$f(\boldsymbol{x} + E_i h) \leq f(\boldsymbol{x}) + \langle \nabla_i f(\boldsymbol{x}),\, h \rangle + \frac{L_i}{2}\|h\|^2, \quad \text{for } i = 1, \ldots, n.$$

BCD – Setup

$$\min f(\boldsymbol{x}) \text{ where } \boldsymbol{x} \in \mathbb{R}^N$$

Assume gradient of block i is Lipschitz continuous**

$$\|\nabla_i f(\boldsymbol{x} + E_i h) - \nabla_i f(\boldsymbol{x})\|_* \leq L_i \|h\|$$

Block gradient $\nabla_i f(\boldsymbol{x})$ is projection of full grad: $E_i^T \nabla f(\boldsymbol{x})$

** — each block can use its own norm

Block Coordinate "Gradient" Descent

- Using the descent lemma, we have blockwise upper bounds

$$f(\boldsymbol{x} + E_i h) \leq f(\boldsymbol{x}) + \langle \nabla_i f(\boldsymbol{x}),\, h \rangle + \tfrac{L_i}{2} \|h\|^2, \quad \text{for } i = 1, \ldots, n.$$

- At each step, minimize these upper bounds!

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$

$$h = -\tfrac{1}{L_i} \nabla_i f(\boldsymbol{x}_k)$$

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \underset{h}{\operatorname{argmin}}\, f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$

$$h = -\tfrac{1}{L_i} \nabla_i f(\boldsymbol{x}_k)$$

- Update the impacted coordinates of $\boldsymbol{x}$, formally

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \underset{h}{\operatorname{argmin}}\, f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$
$$h = -\tfrac{1}{L_i} \nabla_i f(\boldsymbol{x}_k)$$

- Update the impacted coordinates of $\boldsymbol{x}$, formally

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} + h$$
$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)$$

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$
$$h = -\tfrac{1}{L_i} \nabla_i f(\boldsymbol{x}_k)$$

- Update the impacted coordinates of $\boldsymbol{x}$, formally

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} + h$$
$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)$$

Notice: Original BCD had: $x_k^{(i)} = \operatorname{argmin}_h f(\ldots, \underbrace{h}_{\text{block } i}, \ldots)$

Randomized BCD

- For $k \geq 0$ (no init. of $\boldsymbol{x}$ necessary)
- Pick a block i from $[n]$ with probability $p_i > 0$
- Optimize upper bound (partial gradient step) for block i

$$h = \operatorname*{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2} \|h\|^2$$
$$h = -\tfrac{1}{L_i} \nabla_i f(\boldsymbol{x}_k)$$

- Update the impacted coordinates of $\boldsymbol{x}$, formally

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} + h$$
$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)$$

Notice: Original BCD had: $x_k^{(i)} = \operatorname{argmin}_h f(\ldots, \underbrace{h}_{\text{block } i}, \ldots)$

We'll call this BCM (**Block Coordinate Minimization**)

Randomized BCD — slight extension

$$\min f(\boldsymbol{x}) + r(\boldsymbol{x})$$

Randomized BCD — slight extension

$$\min f(\boldsymbol{x}) + r(\boldsymbol{x})$$

- If **block separable** $r(\boldsymbol{x}) := \sum_{i=1}^{n} r_i(x^{(i)})$

Randomized BCD — slight extension

$$\min f(\boldsymbol{x}) + r(\boldsymbol{x})$$

- If **block separable** $r(\boldsymbol{x}) := \sum_{i=1}^{n} r_i(x^{(i)})$

$$x_k^{(i)} = \operatorname*{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h\rangle + \tfrac{L_i}{2}\|h\|^2 + r_i(E_i^T \boldsymbol{x}_k + h)$$

$$x_k^{(i)} = \operatorname{prox}_{r_i}(\cdots)$$

Exercise: Fill in the dots

Randomized BCD — slight extension

$$\min f(\boldsymbol{x}) + r(\boldsymbol{x})$$

- If **block separable** $r(\boldsymbol{x}) := \sum_{i=1}^{n} r_i(x^{(i)})$

$$x_k^{(i)} = \operatorname*{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h\rangle + \tfrac{L_i}{2}\|h\|^2 + r_i(E_i^T \boldsymbol{x}_k + h)$$

$$x_k^{(i)} = \operatorname{prox}_{r_i}(\cdots)$$

Exercise: Fill in the dots

$$h = \operatorname{prox}_{(1/L) r_i}\left(E_i^T \boldsymbol{x}_k - \tfrac{1}{L_i}\nabla_i f(\boldsymbol{x}_k)\right)$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \frac{L_i}{2} \|h\|^2$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + E_i h \\ f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \end{aligned}$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + E_i h \\ f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k) \end{aligned}$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + E_i h \\
f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h \rangle + \tfrac{L_i}{2}\|h\|^2 \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k) \\
f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) - \tfrac{1}{L_i}\|\nabla_i f(\boldsymbol{x}_k)\|^2 + \tfrac{L_i}{2}\left\|-\tfrac{1}{L_i}\nabla_i f(\boldsymbol{x}_k)\right\|^2
\end{aligned}$$

Randomized BCD – analysis

$$h \leftarrow \operatorname{argmin}_h f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h\rangle + \tfrac{L_i}{2}\|h\|^2$$

Descent:

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + E_i h \\
f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) + \langle \nabla_i f(\boldsymbol{x}_k),\, h\rangle + \tfrac{L_i}{2}\|h\|^2 \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k) \\
f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) - \tfrac{1}{L_i}\|\nabla_i f(\boldsymbol{x}_k)\|^2 + \tfrac{L_i}{2}\left\|-\tfrac{1}{L_i}\nabla_i f(\boldsymbol{x}_k)\right\|^2 \\
f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) - \tfrac{1}{2L_i}\|\nabla_i f(\boldsymbol{x}_k)\|^2.
\end{aligned}$$

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \tfrac{1}{2L_i}\|\nabla_i f(\boldsymbol{x}_k)\|^2$$

Randomized BCD – analysis

Expected descent:

$$f(\boldsymbol{x}_k) - \mathbb{E}[f(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)] \quad = \quad \sum_{i=1}^{n} p_i \left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)) \right)$$

Randomized BCD – analysis

Expected descent:

$$
\begin{aligned}
f(\boldsymbol{x}_k) - \mathbb{E}[f(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)] &= \sum_{i=1}^{n} p_i \big(f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)) \big) \\
&\geq \sum_{i=1}^{n} \tfrac{p_i}{2L_i} \|\nabla_i f(\boldsymbol{x}_k)\|^2
\end{aligned}
$$

Randomized BCD – analysis

Expected descent:

$$
\begin{aligned}
f(\boldsymbol{x}_k) - \mathbb{E}[f(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)] &= \sum_{i=1}^{n} p_i \big(f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)) \big) \\
&\geq \sum_{i=1}^{n} \tfrac{p_i}{2L_i} \|\nabla_i f(\boldsymbol{x}_k)\|^2 \\
&= \tfrac{1}{2} \|\nabla f(\boldsymbol{x}_k)\|_W^2 \quad \text{(suitable } W\text{)}.
\end{aligned}
$$

Randomized BCD – analysis

Expected descent:

$$\begin{aligned} f(\boldsymbol{x}_k) - \mathbb{E}[f(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)] &= \sum_{i=1}^{n} p_i \big(f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k)) \big) \\ &\geq \sum_{i=1}^{n} \tfrac{p_i}{2L_i} \|\nabla_i f(\boldsymbol{x}_k)\|^2 \\ &= \tfrac{1}{2} \|\nabla f(\boldsymbol{x}_k)\|_W^2 \quad \text{(suitable } W\text{)}. \end{aligned}$$

Exercise: What is the expected descent with uniform probabilities?

Randomized BCD – analysis

Expected descent:

$$\begin{aligned} f(\boldsymbol{x}_k) - \mathbb{E}[f(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)] &= \sum_{i=1}^{n} p_i \big(f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k - \tfrac{1}{L_i} E_i \nabla_i f(\boldsymbol{x}_k))\big) \\ &\geq \sum_{i=1}^{n} \tfrac{p_i}{2L_i} \|\nabla_i f(\boldsymbol{x}_k)\|^2 \\ &= \tfrac{1}{2}\|\nabla f(\boldsymbol{x}_k)\|_W^2 \quad \text{(suitable } W\text{)}. \end{aligned}$$

Exercise: What is the expected descent with uniform probabilities?

Descent combined with some more notation and hard work yields

$$O(\tfrac{n}{\epsilon} \sum\nolimits_i L_i \|x_0^{(i)} - x_*^{(i)}\|^2)$$

as the iteration complexity of obtaining $\mathbb{E}[f(\boldsymbol{x}_k)] - f^* \leq \epsilon$

Exercise

- Recall Lasso problem: $\min \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1$
- Here $x \in \mathbb{R}^N$
- Make $n = N$ blocks
- Show what the Randomized BCD iterations look like
- Notice, 1D prox operations for $\lambda|\cdot|$ arise
- Try to implement it as efficiently as you can (i.e., do not copy or update vectors / coordinates than necessary)

Exercise – details

Assuming $n = N$ blocks, each update is scalar valued.

- Let $x_0 = 0$; $y_0 = Ax_0 - b = -b$
- For $k \geq 0$
  - Pick random coordinate j
  - Compute $\alpha \leftarrow \langle a_j, y \rangle$ – i.e., $\nabla_j f(\boldsymbol{x}_k)$
  - Min $\alpha h + \frac{L_i}{2} h^2 + \lambda |h|$

$$h = \text{prox}_{\lambda|\cdot|}(x_j - \tfrac{1}{L_j}\alpha)$$

$$h = \text{sgn}(x_j - \tfrac{1}{L_j}\alpha)\max(|x_j - \tfrac{1}{L_j}\alpha| - \lambda, 0)$$

Exercise – details

Assuming $n = N$ blocks, each update is scalar valued.

- Let $x_0 = 0$; $y_0 = Ax_0 - b = -b$
- For $k \geq 0$
  - Pick random coordinate j
  - Compute $\alpha \leftarrow \langle a_j, y \rangle$ – i.e., $\nabla_j f(\boldsymbol{x}_k)$
  - Min $\alpha h + \frac{L_i}{2} h^2 + \lambda |h|$

$$h = \text{prox}_{\lambda|\cdot|}(x_j - \tfrac{1}{L_j}\alpha)$$

$$h = \text{sgn}(x_j - \tfrac{1}{L_j}\alpha)\max(|x_j - \tfrac{1}{L_j}\alpha| - \lambda, 0)$$

  - Update: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\boldsymbol{e}_j$

Exercise – details

Assuming $n = N$ blocks, each update is scalar valued.

- Let $x_0 = 0$; $y_0 = Ax_0 - b = -b$
- For $k \geq 0$
  - Pick random coordinate j
  - Compute $\alpha \leftarrow \langle a_j, y \rangle$ – i.e., $\nabla_j f(\boldsymbol{x}_k)$
  - Min $\alpha h + \frac{L_i}{2} h^2 + \lambda |h|$

$$h = \text{prox}_{\lambda|\cdot|}(x_j - \tfrac{1}{L_j}\alpha)$$

$$h = \text{sgn}(x_j - \tfrac{1}{L_j}\alpha)\max(|x_j - \tfrac{1}{L_j}\alpha| - \lambda, 0)$$

  - Update: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\boldsymbol{e}_j$
  - Update: $y_{k+1} \leftarrow y_k + ha_j$

Parallel BCD

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_n)$$

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_n)$$

What if?

$$\min f(x) = \sum_i f_i(x_i)$$

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_n)$$

What if?

$$\min f(x) = \sum_i f_i(x_i)$$

- Can solve all n problems **independently** in **parallel**
- In theory: n times speedup possible compared to serial case

Parallel BCD

Previously

$$\min f(x) = f(x_1, \ldots, x_n)$$

What if?

$$\min f(x) = \sum_i f_i(x_i)$$

- Can solve all n problems **independently** in **parallel**
- In theory: n times speedup possible compared to serial case
- So if objective functions are "almost separable" we would still expect high speedup, diminished by amount of **separability**
- Big data problems often have this "almost separable" structure!

Partial Separability

Consider the **sparse** data matrix

$$\begin{pmatrix} d_{11} & d_{12} & & & \\ & d_{22} & d_{23} & & \\ & & \ddots & \ddots & \end{pmatrix} \in \mathbb{R}^{m \times n},$$

Partial Separability

Consider the **sparse** data matrix

$$\begin{pmatrix} d_{11} & d_{12} & & \\ & d_{22} & d_{23} & \\ & & \ddots & \ddots \end{pmatrix} \in \mathbb{R}^{m \times n},$$

- Objective $f(x) = \|Dx - b\|_2^2 = \sum_{i=1}^m (d_i^T x - b_i)^2$ also equals
$$(d_{11}x_1 + d_{12}x_2 - b_1)^2 + (d_{22}x_2 + d_{23}x_3 - b_2)^2 + \cdots$$
- Each term depends on only 2 coordinates
- Formally, we could write this as
$$f(x) = \sum\nolimits_{J \in \mathscr{J}} f_J(x),$$
where $\mathscr{J} = \{\{1,2\},\{2,3\},\cdots\}$
- Key point: $f_J(x)$ depends only on x_j for $j \in J$.

Partial Separability

$$\min f(x) \text{ s.t. } x \in \mathbb{R}^n$$

Def. Let $\mathscr{J}$ be a collection of subsets of $\{1, \ldots, n\}$. We say f is **partially separable of degree ω** if it can be written as

$$f(x) = \sum_{J \in \mathscr{J}} f_J(x),$$

where each f_J depends only on x_j for $j \in J$, and

$$|J| \leq \omega \quad \forall J \in \mathscr{J}.$$

Example: If $D_{m \times n}$ is a sparse matrix, then $\omega = \max_{1 \leq i \leq m} \|d_i^T\|_0$

Partial Separability

$$\min f(x) \text{ s.t. } x \in \mathbb{R}^n$$

Def. Let $\mathscr{J}$ be a collection of subsets of $\{1, \ldots, n\}$. We say f is **partially separable of degree ω** if it can be written as

$$f(x) = \sum_{J \in \mathscr{J}} f_J(x),$$

where each f_J depends only on x_j for $j \in J$, and

$$|J| \leq \omega \quad \forall J \in \mathscr{J}.$$

Example: If $D_{m \times n}$ is a sparse matrix, then $\omega = \max_{1 \leq i \leq m} \|d_i^T\|_0$

Exercise: Extend this notion to $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(n)})$

Hint: Now, f_J will depend only on $x^{(j)}$ for $j \in J$

Parallel Stochastic Gradient!

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, n\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

Parallel Stochastic Gradient!

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, n\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

- **Atomic update** only for $x_j \leftarrow x_j - a$ (not for gradient)

Parallel Stochastic Gradient!

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, n\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

- **Atomic update** only for $x_j \leftarrow x_j - a$ (not for gradient)
- Since the actual coordinate j can arise in various J, processors can overwrite each others' work.

Parallel Stochastic Gradient!

Each core runs the computation:

1. Sample coordinates J from $\{1, \ldots, n\}$ (all sets of variables)
2. Read current state of x_J from shared memory
3. For each individual coordinate $j \in J$
   $x_j \leftarrow x_j - \alpha_k [\nabla f_J(x_J)]_j$

- **Atomic update** only for $x_j \leftarrow x_j - a$ (not for gradient)
- Since the actual coordinate j can arise in various J, processors can overwrite each others' work.
- But if **partial overlaps (separability)**, coordinate j does not appear in too many different subsets J, method works fine!

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, n\}$

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, n\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(\boldsymbol{x}_k)$$

$\longrightarrow$ w_i typically L_i; β depends on degree of separability ω

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, n\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(\boldsymbol{x}_k)$$

$\longrightarrow$ w_i typically L_i; β depends on degree of separability ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, n\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(\boldsymbol{x}_k)$$

$\longrightarrow$ w_i typically L_i; β depends on degree of separability ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees
- ♠ Theory requires **atomic updates**

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, n\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(\boldsymbol{x}_k)$$

$\longrightarrow$ w_i typically L_i; β depends on degree of separability ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees
- ♠ Theory requires **atomic updates**
- ♠ Useful to implement **asynchronously** (i.e., use whatever latest $x^{(i)}$ a given core has access to)

Parallel BCD

1. Choose initial point $\boldsymbol{x}_0 \in \mathbb{R}^N$
2. For $k \geq 0$
   - Randomly pick (in parallel) a set of blocks $S_k \subset \{1, \ldots, n\}$
   - Perform BCD updates (in parallel) for $i \in S_k$

$$\boldsymbol{x}_{k+1}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} - \frac{1}{\beta w_i} \nabla_i f(\boldsymbol{x}_k)$$

$\longrightarrow$ w_i typically L_i; β depends on degree of separability ω

- ♠ Uniform sampling of blocks (or just coordinates)
- ♠ More careful sampling leads to better guarantees
- ♠ Theory requires **atomic updates**
- ♠ Useful to implement **asynchronously** (i.e., use whatever latest $x^{(i)}$ a given core has access to)
- ♠ Theory of above method requires **guaranteed descent**

ADMM & Co.

Background

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

Background

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

Typical approach:

- Form the *Lagrangian*: $L(x, y) = f(x) + y^T(Ax - b)$

Background

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

Typical approach:

- Form the *Lagrangian*: $L(x, y) = f(x) + y^T(Ax - b)$
- Compute **dual function**

$$g(y) := \min_x \; L(x, y)$$

Background

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

Typical approach:

- Form the *Lagrangian*: $L(x, y) = f(x) + y^T(Ax - b)$
- Compute **dual function**

$$g(y) := \min_x \; L(x, y)$$

- Solve dual problem: $\max_y g(y)$ to get y^*

Background

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

Typical approach:

- Form the *Lagrangian*: $L(x, y) = f(x) + y^T(Ax - b)$
- Compute **dual function**

$$g(y) := \min_x \; L(x, y)$$

- Solve dual problem: $\max_y g(y)$ to get y^*
- Recover primal solution: $x^* = \operatorname{argmin} L(x, y^*)$

How to solve dual?

Use some gradient method on dual!

How to solve dual?

Use some gradient method on dual!

$$y_{k+1} = y_k + \alpha_k \nabla g(y_k)$$

(notice $+\alpha_k$ since we are doing ascent)

How to solve dual?

Use some gradient method on dual!

$$y_{k+1} = y_k + \alpha_k \nabla g(y_k)$$

(notice $+\alpha_k$ since we are doing ascent)

But what is $\nabla g(y)$?

How to solve dual?

Use some gradient method on dual!

$$y_{k+1} = y_k + \alpha_k \nabla g(y_k)$$

(notice $+\alpha_k$ since we are doing ascent)

But what is $\nabla g(y)$?

$$\begin{aligned} g(y) &= \min_x f(x) + y^T(Ax - b) \\ \nabla g(y_k) &= A\bar{x} - b \\ \bar{x} &= \operatorname*{argmin}_x L(x, y_k) \end{aligned}$$

Dual ascent method

$$
\begin{aligned}
x_{k+1} &= \operatorname{argmin} L(x, y_k) \\
y_{k+1} &= y_k + \alpha_k (A x_{k+1} - b)
\end{aligned}
$$

Dual ascent method

$$\begin{aligned} x_{k+1} &= \operatorname{argmin} L(x, y_k) \\ y_{k+1} &= y_k + \alpha_k (A x_{k+1} - b) \end{aligned}$$

Works, but expensive; needs strong technical assumptions on $f(x)$

Dual ascent method

$$\begin{aligned} x_{k+1} &= \operatorname{argmin} L(x, y_k) \\ y_{k+1} &= y_k + \alpha_k (A x_{k+1} - b) \end{aligned}$$

Works, but expensive; needs strong technical assumptions on $f(x)$

What if **fully separable** f

$$f(x) = \sum_i f_i(x_i)$$

Dual ascent – fully separable

For fully separable f, the Lagrangian is also fully separable

$$L(x, y) = \sum_i (f_i(x_i) + y^T A_i x_i) - y^T b$$

Dual ascent – fully separable

For fully separable f, the Lagrangian is also fully separable

$$L(x, y) = \sum_i (f_i(x_i) + y^T A_i x_i) - y^T b$$

Thus, $\operatorname{argmin} L(x, y_k)$ splits into n *separate minimizations*

$$(x_i)_{k+1} = \operatorname*{argmin}_{x_i} (f_i(x_i) + y^T A_i x_i)$$

All can be done in parallel

Dual decomposition

The above idea leads to *dual decomposition*—classic idea from the 60s (Everett, Danzig, Wolfe, Benders, ...)

Dual decomposition

The above idea leads to *dual decomposition*—classic idea from the 60s (Everett, Danzig, Wolfe, Benders, ...)

$$
\begin{aligned}
[x_i]_{k+1} &= \operatorname*{argmin}_{x_i}(f_i(x_i) + y^T A_i x_i) \quad i = 1, \ldots, n \\
y_{k+1} &= y_k + \alpha_k(\sum_{i=1}^{n} A_i [x_i]_{k+1} - b)
\end{aligned}
$$

Dual decomposition

The above idea leads to *dual decomposition*—classic idea from the 60s (Everett, Danzig, Wolfe, Benders, ...)

$$
\begin{aligned}
[x_i]_{k+1} &= \operatorname*{argmin}_{x_i}(f_i(x_i) + y^T A_i x_i) \quad i = 1, \ldots, n \\
y_{k+1} &= y_k + \alpha_k(\sum_{i=1}^{n} A_i [x_i]_{k+1} - b)
\end{aligned}
$$

distributed processing

- distribute y_k

Dual decomposition

The above idea leads to *dual decomposition*—classic idea from the 60s (Everett, Danzig, Wolfe, Benders, ...)

$$
\begin{aligned}
[x_i]_{k+1} &= \operatorname*{argmin}_{x_i}(f_i(x_i) + y^T A_i x_i) \quad i = 1, \ldots, n \\
y_{k+1} &= y_k + \alpha_k(\sum_{i=1}^{n} A_i [x_i]_{k+1} - b)
\end{aligned}
$$

distributed processing

- distribute y_k
- compute $(x_i)_{k+1}$ (simultaneously)

Dual decomposition

The above idea leads to *dual decomposition*—classic idea from the 60s (Everett, Danzig, Wolfe, Benders, ...)

$$
\begin{aligned}
[x_i]_{k+1} &= \operatorname*{argmin}_{x_i}(f_i(x_i) + y^T A_i x_i) \quad i = 1, \ldots, n \\
y_{k+1} &= y_k + \alpha_k(\sum_{i=1}^{n} A_i[x_i]_{k+1} - b)
\end{aligned}
$$

distributed processing

- distribute y_k
- compute $(x_i)_{k+1}$ (simultaneously)
- collect updated values $A_i(x_i)_{k+1}$

Dual decomposition

The above idea leads to *dual decomposition*—classic idea from the 60s (Everett, Danzig, Wolfe, Benders, ...)

$$
\begin{aligned}
[x_i]_{k+1} &= \operatorname*{argmin}_{x_i}(f_i(x_i) + y^T A_i x_i) \quad i = 1, \ldots, n \\
y_{k+1} &= y_k + \alpha_k(\sum_{i=1}^{n} A_i[x_i]_{k+1} - b)
\end{aligned}
$$

distributed processing

- distribute y_k
- compute $(x_i)_{k+1}$ (simultaneously)
- collect updated values $A_i(x_i)_{k+1}$
- centralize to compute y_{k+1}

This method works but can be often very slow.

Next time

- ADMM for distributed computation
- Basic methods in distributed optimization

References

- P. Richtárik, M. Takáč. *Parallel Coordniate Descent Methods for Big Data Optimization* (Dec. 2012)
- F. Niu, et al. *Hogwild!: A lock-free approach to parallelizing stochastic gradient descent*
- S. P. Boyd. Slides on ADMM.